Oracle Database 12c Release 2 and 18c New Indexing Features

Richard Foote Consulting

Richard Foote

- Working in IT for 30+ years , 20+ years with Oracle Database
- 19 years employed in Australian Federal Government in various IT roles
- Worked for Oracle Corporation between 1996 and 2002 and between 2011 and 2017
- In September 2017, started my own independent company Richard Foote Consulting
- I've been responsible for many large scale, mission critical, "life-dependant" classified Oracle systems, tuned numerous databases often with 10x performance improvements
- Oracle OakTable Member since 2002 and awarded Oracle ACE Director in 2008
- Regular speaker at user group meetings and conferences such as Oracle OpenWorld, IOUG Collaborate, Hotsos Symposium, AUSOUG InSync, ODTUG Kscope, UKOUG Tech Conference, E4 Enkitec Extreme Exadata Expo, Trivadis Performance Days…
- Richard Foote's Oracle Blog: https://richardfoote.wordpress.com
- Richard Foote Consulting: https//richardfooteconsulting.com
- Spend as much free time as possible listening to the music of David Bowie !!

richardfoote.wordpress.com

Oracle Indexing Internals and Best Practices 2 Day Seminar

Webinars Coming In March 2019 !!

Of benefit to DBAs, Developers, Solution Architects and anyone else interested in designing, developing or maintaining high performance Oracle-based applications/databases.

Indexes fundamental to every Oracle database and crucial for optimal performance. Many applications/databases are suboptimal and run inefficiently primarily because of inappropriate indexing strategies.

This highly acclaimed webinar covers many useful tips and strategies to maximise benefits of indexes on application/database performance and scalability, as well as in maximising Oracle database investments.

4-8 March & 26-30 March 2019 Limited Places Available !!

richardfooteconsulting.com/indexing-webinar

12c Release 2 and 18c: New Indexing Related Features

- Long Identifiers
- High Advanced Index Compression
- Tracking Index Usage
- Online Table Move (and Automatic Index Maintenance)
- Online Conversion to Partitioned Table (and Partitioned Indexes)
- Online Modify Partitioned Table To Partitioned Table (new 18c)
- Deferred Invalidation of Cursors During Index Creation/Rebuild
- Indexing Case Sensitive Database
- JSON Search Index (including new 18c features)
- Memoptimized Rowstore (new 18c)
- Scalable Sequences (new 18c)
- Oracle Database 19c Indexing Related Surprise !!

Identifiers – Pre 12.2

```
SQL> create table bowie (id number, code number, name varchar2(42));

Table created.

SQL> insert into bowie select rownum, mod(rownum,10), 'DAVID BOWIE'
     from dual connect by level <= 1000000;

1000000 rows created.

SQL> commit;

Commit complete.

SQL> create index
this_index_will_be_used_to_get_data_from_the_bowie_table_in_scenarios_when_the_code_column_is_used_in_
predicates_idx on bowie(code);
create index this_index_will_be_used_to_get_data_from_the_bowie_table_in_scenari
os_when_the_code_column_is_used_in_predicates_idx on bowie(code)
            *
ERROR at line 1:
ORA-00972: identifier is too long
```

Maximum length of identifiers 30 characters

New in 12.2 Long Identifiers: 128 Characters

```
SQL> create index
this_index_will_be_used_to_get_data_from_the_bowie_table_in_scenarios_when_the_code_column_is_used_in_
predicates_idx on bowie(code);

Index created.

SQL> select index_name, leaf_blocks, status from dba_indexes where table_name='BOWIE';

INDEX_NAME                                          LEAF_BLOCKS STATUS
--------------------------------------------------- ----------- --------
THIS_INDEX_WILL_BE_USED_TO_GET_DATA_FROM_THE_BOWIE         1939 VALID
_TABLE_IN_SCENARIOS_WHEN_THE_CODE_COLUMN_IS_USED_I
N_PREDICATES_IDX

SQL> alter table bowie add constraint
the_primary_key_of_the_bowie_table_is_the_id_column_so_please_stop_trying_to_insert_a_duplicate_id_val
ue_dumbo primary key (id);

Table altered.

SQL> insert into bowie values (42, 42, 'David Bowie');
insert into bowie values (42, 42, 'David Bowie')
*
ERROR at line 1:
ORA-00001: unique constraint
(BOWIE.THE_PRIMARY_KEY_OF_THE_BOWIE_TABLE_IS_THE_ID_COLUMN_SO_PLEASE_STOP_TRYING
_TO_INSERT_A_DUPLICATE_ID_VALUE_DUMBO) violated
```

New in 12.2: Index High Advanced Compression

```
SQL> create table bowie (id number, code number, name varchar2(42));
Table created.
SQL> insert into bowie select rownum, rownum, 'ZIGGY STARDUST' from dual connect by level <= 1000000;
1000000 rows created.
SQL> update bowie set code = 42 where id between 250000 and 499999;
250000 rows updated.
SQL> commit;
Commit complete.
SQL> create index bowie_code_i on bowie(code) nocompress;
Index created.
SQL> select index_name, leaf_blocks, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS COMPRESSION
------------ ----------- -------------
BOWIE_CODE_I        2157 DISABLED
```

Index High Advanced Compression

```
SQL> alter index bowie_code_i rebuild compress;

Index altered.

SQL> select index_name, leaf_blocks, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS COMPRESSION
------------ ----------- -------------
BOWIE_CODE_I        2684 ENABLED        => Increased by 526 Leaf Blocks

SQL> alter index bowie_code_i rebuild compress advanced low;

Index altered.

SQL> select index_name, leaf_blocks, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS COMPRESSION
------------ ----------- -------------
BOWIE_CODE_I        2054 ADVANCED LOW    => Reduced by 103 Leaf Blocks
```

Index High Advanced Compression

```
SQL> alter index bowie_code_i rebuild compress advanced high;

Index altered.

SQL> select index_name, leaf_blocks, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS COMPRESSION
------------ ----------- -------------
BOWIE_CODE_I           0 ADVANCED HIGH

--- This was due to bug 22094934 in first 12.2 release

SQL> exec dbms_stats.gather_index_stats(ownname=>null, indname=>'BOWIE_CODE_I');

PL/SQL procedure successfully completed.

SQL> select index_name, leaf_blocks, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS COMPRESSION
------------ ----------- -------------
BOWIE_CODE_I         758 ADVANCED HIGH    => Reduced now by a massive 1399 Leaf Blocks !!
```

Index High Advanced Compression

```
SQL> create table bowie (id number, code number, name varchar2(42));
Table created.
SQL> insert into bowie select rownum, mod(rownum,10), 'DAVID BOWIE' from dual connect by level <= 1000000;
1000000 rows created.
SQL> commit;
Commit complete.
SQL> create index bowie_idx on bowie(code, id) pctfree 0;
Index created.
SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where  table_name='BOWIE';
INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           2361          2 DISABLED
SQL> alter index bowie_idx rebuild compress;
Index altered.
SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';
INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           3120          2 ENABLED    => Up by 759 leaf blocks as most index entries unique
```

Index High Advanced Compression

```
SQL> alter index bowie_idx rebuild compress 1;
Index altered.
SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';
INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           2002          2 ENABLED  => Decreases by 359 leaf blocks due to replicated code values
SQL> alter index bowie_idx rebuild compress advanced low;
Index altered.
SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           2002          2 ADVANCED LOW  => Same as compress 1, but performed automatically
SQL> alter index bowie_idx rebuild compress advanced high;
Index altered.
SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX            895          2 ADVANCED HIGH => Decreases by 1466 leaf blocks as both columns compressed
```

Index High Advanced Compression

```
SQL> create index bowie_idx on bowie(id, code) pctfree 0;

Index created.

SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           2363          2 DISABLED

SQL> alter index bowie_idx rebuild compress;

Index altered.

SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           3115          2 ENABLED    => Increases by 752 leaf blocks as again index effectively
                                                   unique
```

Index High Advanced Compression

```
SQL> alter index bowie_idx rebuild compress 1;

Index altered.

SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS    BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           3115          2 ENABLED    => Just as bad as compress as leading column unique

SQL> alter index bowie_idx rebuild compress advanced low;

Index altered.

SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS    BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           2363          2 ADVANCED LOW => No difference to nocompress, but at least no larger

SQL> alter index bowie_idx rebuild compress advanced high;

Index altered.

SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where  table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS    BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_IDX           1057          2 ADVANCED HIGH  => decrease ½ the size, not that diff to index with code
                                                     leading
```

Index High Advanced Compression

Example with a single column, Unique Index:

```
SQL> create table bowie (id number, code number, name varchar2(42));

Table created.

SQL> insert into bowie select rownum, rownum, 'ZIGGY STARDUST' from dual connect by level <= 1000000;

1000000 rows created.

SQL> commit;

Commit complete.

SQL> create unique index bowie_id_i on bowie(id);

Index created.

SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_ID_I          2087          2 DISABLED
```

Index High Advanced Compression

```
SQL> alter index bowie_id_i rebuild compress;
alter index bowie_id_i rebuild compress
*
ERROR at line 1:
ORA-25193: cannot use COMPRESS option for a single column key

SQL> alter index bowie_id_i rebuild compress advanced low;
alter index bowie_id_i rebuild compress advanced low
*
ERROR at line 1:
ORA-25193: cannot use COMPRESS option for a single column key

SQL> alter index bowie_id_i rebuild compress advanced high;

Index altered.

SQL> select index_name, leaf_blocks, blevel, compression from user_indexes where table_name='BOWIE';

INDEX_NAME   LEAF_BLOCKS     BLEVEL COMPRESSION
------------ ----------- ---------- -------------
BOWIE_ID_I           925          2 ADVANCED HIGH => Reduced by more than 50%, down from 2087 leaf blocks
```

Index Advanced Compression – Default

```
SQL> create table bowie (id number, code number, name varchar2(42));

Table created.

SQL> insert into bowie select rownum, mod(rownum,10), 'DAVID BOWIE'
     from dual connect by level <= 1000000;

1000000 rows created.

SQL> commit;

Commit complete.

SQL> create index bowie_code_i on bowie(code) tablespace bowie nocompress;

Index created.

SQL> select index_name, tablespace_name, leaf_blocks, compression
     from dba_indexes where index_name='BOWIE_CODE_I';

INDEX_NAME      TABLESPACE_NAME                LEAF_BLOCKS COMPRESSION
--------------- ------------------------------ ----------- -------------
BOWIE_CODE_I    BOWIE                                 1939 DISABLED
```

Index Advanced Compression – Default

```
SQL> drop index bowie_code_i;

Index dropped.

SQL> create index bowie_code_i on bowie(code) tablespace bowie;

Index created.

SQL> exec dbms_stats.gather_index_stats(ownname=>null, indname=>'BOWIE_CODE_I');

PL/SQL procedure successfully completed.

SQL> select index_name, tablespace_name, leaf_blocks, compression from dba_indexes
     where index_name='BOWIE_CODE_I';

INDEX_NAME      TABLESPACE_NAME                LEAF_BLOCKS COMPRESSION
--------------- ------------------------------ ----------- -------------
BOWIE_CODE_I    BOWIE                                  353 ADVANCED HIGH
```

How has the index been created with Index Advanced Compression ?

Index Advanced Compression – Default

```
SQL> alter system set db_index_compression_inheritance=tablespace scope=both;

System altered.

SQL> alter tablespace bowie default index compress advanced high;

Tablespace altered.

SQL> select tablespace_name, def_index_compression, index_compress_for
     from dba_tablespaces where tablespace_name = 'BOWIE';

TABLESPACE_NAME                DEF_INDE INDEX_COMPRES
------------------------------ -------- -------------
BOWIE                          ENABLED  ADVANCED HIGH
```

Can be set by default at the Tablespace or Table level

Index High Advanced Compression

Note: DML overheads associated with Index High Advanced Compression:

```
SQL> create table bowie (id number, code number, name varchar2(42));

Table created.

SQL> create index bowie_code_i on bowie(code) nocompress;

Index created.

SQL> insert into bowie select rownum, rownum, 'ZIGGY STARDUST' from dual connect by level <= 1000000;

1000000 rows created.
Elapsed: 00:00:02.09

SQL> commit;

Commit complete.

SQL> update bowie set code = 42 where id between 250000 and 499999;

250000 rows updated.
Elapsed: 00:00:07.91
```

Index High Advanced Compression

```
SQL> create table bowie (id number, code number, name varchar2(42));

Table created.

SQL> create index bowie_code_idx on bowie(code) compress advanced high;

Index created.

SQL> insert into bowie select rownum, rownum, 'ZIGGY STARDUST' from dual connect  by level <= 1000000;

1000000 rows created.
Elapsed: 00:00:08.16

SQL> commit;

Commit complete.

SQL> update bowie set code = 42 where id between 250000 and 499999;

250000 rows updated.
Elapsed: 00:00:10.09
```

Greater CPU consumption and overall response times.
Caution setting Index High Compression on tables with high DML workloads.

Index Monitoring

```
SQL> create table bowie (id number, code number, name varchar2(42));
Table created.
SQL> insert into bowie select rownum, mod(rownum,10000), 'DAVID BOWIE' from dual connect by level <= 1000000;
1000000 rows created.
SQL> commit;
Commit complete.
SQL> create index bowie_code_idx on bowie(code);
Index created.
SQL> alter index bowie_code_idx monitoring usage;
Index altered.
SQL> select * from  bowie where code=42;
100 rows selected.
SQL> select * from v$object_usage where index_name = 'BOWIE_CODE_IDX';
INDEX_NAME      TABLE_NAME      MON USE START_MONITORING    END_MONITORING
--------------- --------------- --- --- ------------------- -------------------
BOWIE_CODE_IDX  BOWIE           YES YES 11/29/2017 05:49:34                  => But how often is it used?
```

New in 12.2 :Tracking Index Usage

```
SQL> create table radiohead (id number, code number, name varchar2(42));

Table created.

SQL> insert into radiohead select rownum, mod(rownum,10000), 'OK COMPUTER'
     from dual connect by level <= 1000000;

1000000 rows created.

SQL> commit;

Commit complete.

SQL> create index radiohead_code_i on radiohead(code);

Index created.

SQL> create unique index radiohead_id_i on radiohead(id);

Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'RADIOHEAD');

PL/SQL procedure successfully completed.
```

Tracking Index Usage

```
SQL> select active_elem_count, alloc_elem_count, max_elem_count from v$index_usage_info;

ACTIVE_ELEM_COUNT ALLOC_ELEM_COUNT MAX_ELEM_COUNT
----------------- ---------------- --------------
                0               42          30000

SQL> select name, total_access_count, total_exec_count, total_rows_returned
     from dba_index_usage where owner='RADIOHEAD';

no rows selected

SQL> select * from radiohead where id=42;    ---- Execute 10 times

SQL> select * from radiohead where code=42;  ---- Execute 10 times

SQL> select active_elem_count, alloc_elem_count, max_elem_count from v$index_usage_info;

ACTIVE_ELEM_COUNT ALLOC_ELEM_COUNT MAX_ELEM_COUNT
----------------- ---------------- --------------
                1               43          30000  ➔ Only one index has been picked up
```

Unique index usage with equality predicates are not captured

Tracking Index Usage

```
SQL> desc dba_index_usage
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OBJECT_ID                                 NOT NULL NUMBER
 NAME                                      NOT NULL VARCHAR2(128)
 OWNER                                     NOT NULL VARCHAR2(128)
 TOTAL_ACCESS_COUNT                                 NUMBER
 TOTAL_EXEC_COUNT                                   NUMBER
 TOTAL_ROWS_RETURNED                                NUMBER
 BUCKET_0_ACCESS_COUNT                              NUMBER
 BUCKET_1_ACCESS_COUNT                              NUMBER
 BUCKET_2_10_ACCESS_COUNT                           NUMBER
 BUCKET_2_10_ROWS_RETURNED                          NUMBER
 BUCKET_11_100_ACCESS_COUNT                         NUMBER
 BUCKET_11_100_ROWS_RETURNED                        NUMBER
 BUCKET_101_1000_ACCESS_COUNT                       NUMBER
 BUCKET_101_1000_ROWS_RETURNED                      NUMBER
 BUCKET_1000_PLUS_ACCESS_COUNT                      NUMBER
 BUCKET_1000_PLUS_ROWS_RETURNED                     NUMBER
 LAST_USED                                          DATE

SQL> select name, total_exec_count, total_rows_returned, last_used from dba_index_usage
     where name like 'RADIOHEAD%';

NAME                 TOTAL_EXEC_COUNT TOTAL_ROWS_RETURNED LAST_USED
-------------------- ---------------- -------------------- ---------
RADIOHEAD_CODE_I                    8                  800 11-JUN-18
```

➔ Not all occurrences picked up

Tracking Index Usage

```
SQL> select * from radiohead where id between 42 and 420;     ---- Execute 10 times
379 rows selected.

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                |   380 |  7980 |     5   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| RADIOHEAD      |   380 |  7980 |     5   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | RADIOHEAD_ID_I |   380 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

SQL> select active_elem_count, alloc_elem_count, max_elem_count from v$index_usage_info;

ACTIVE_ELEM_COUNT ALLOC_ELEM_COUNT MAX_ELEM_COUNT
----------------- ---------------- --------------
                1               44          30000

SQL> select name, total_exec_count, total_rows_returned, last_used from dba_index_usage
     where name like 'RADIOHEAD%';

NAME                 TOTAL_EXEC_COUNT TOTAL_ROWS_RETURNED LAST_USED
-------------------- ---------------- ------------------- ---------
RADIOHEAD_CODE_I                    8                 800 11-JUN-18
RADIOHEAD_ID_I                     10                3790 11-JUN-18 ➔ Unique indexes captured for non
                                                                      equality predicates
```

Tracking Index Usage

```
--- By default, the maximum index entries that can be tracked is 30000;
however it can be increased with:

alter system set “_iux_max_entries=50000”;

--- To collect index usage stats for all the sql executions you can set the
following parameter:

alter system set "_iut_stat_collection_type"=ALL;

--- For sampled, which is default you can set it as:

alter system set "_iut_stat_collection_type"=SAMPLED;
```

Note: Has same rules regarding not capturing implicit usage of FK indexes and index statistics as with Index Monitoring

Tracking Index Usage: FK Indexes

```
SQL> create table daddy (id number constraint daddy_pk primary key, name varchar2(42));
Table created.
SQL> insert into daddy select rownum, 'DAVID BOWIE' from dual connect by level <= 100;
100 rows created.
SQL> commit;
Commit complete.
SQL> create table kiddie (id number, name varchar2(20), fk number, constraint kiddie_fk foreign key(fk)
references daddy(id));
Table created.
SQL> insert into kiddie select rownum, 'MAJOR TOM', 1 from dual connect by level <= 1000000;
1000000 rows created.
SQL> commit;
Commit complete.
SQL> create index kiddie_fk_i on kiddie(fk);
Index created.
SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'DADDY');
PL/SQL procedure successfully completed.
SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'KIDDIE');
PL/SQL procedure successfully completed.
```

Tracking Index Usage: FK Indexes

```
SQL> select active_elem_count, alloc_elem_count, max_elem_count from v$index_usage_info;
ACTIVE_ELEM_COUNT ALLOC_ELEM_COUNT MAX_ELEM_COUNT
                0               46          30000
SQL> delete daddy where id = 2;
1 row deleted.
Statistics
----------------------------------------------------------
         40  recursive calls
         13  db block gets
         42  consistent gets
          0  physical reads
       1824  redo size
        581  bytes sent via SQL*Net to client
        758  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          5  sorts (memory)
          0  sorts (disk)
          1  rows processed
SQL> select active_elem_count, alloc_elem_count, max_elem_count from v$index_usage_info;
ACTIVE_ELEM_COUNT ALLOC_ELEM_COUNT MAX_ELEM_COUNT
                0               46          30000
```

Tracking Index Usage: FK Indexes

```
SQL> delete daddy where id = 3;
1 row deleted.
SQL> delete daddy where id = 4;
1 row deleted.
SQL> delete daddy where id = 5;
1 row deleted.
SQL> delete daddy where id = 6;
1 row deleted.
SQL> delete daddy where id = 7;
1 row deleted.
SQL> delete daddy where id = 8;
1 row deleted.
SQL> delete daddy where id = 9;
1 row deleted.
SQL> delete daddy where id = 10;
1 row deleted.

SQL> select active_elem_count, alloc_elem_count, max_elem_count from v$index_usage_info;
ACTIVE_ELEM_COUNT ALLOC_ELEM_COUNT MAX_ELEM_COUNT
                0               46          30000
```

Tracking Index Usage: FK Indexes

```
SQL> select name, total_exec_count, total_rows_returned, last_used from dba_index_usage
     where name = 'KIDDIE_FK_I';

no rows selected

SQL> drop index kiddie_fk_i;

Index dropped.

SQL> delete daddy where id = 11;

1 row deleted.

Statistics
----------------------------------------------------------
         25  recursive calls
          6  db block gets
       3254  consistent gets
          0  physical reads
        812  redo size
        581  bytes sent via SQL*Net to client
        759  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          6  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Table Move (and Index Maintenance)

Before 12.2

```
SQL> create table bowie (id number, code number, name varchar2(42));

Table created.

SQL> insert into bowie select rownum, mod(rownum,10), 'DAVID BOWIE'
     from dual connect by level <= 1000000;

1000000 rows created.

SQL> commit;

Commit complete.

SQL> create index bowie_code_idx on bowie(code);

Index created.

SQL> select index_name, leaf_blocks, status from dba_indexes where table_name='BOWIE';

INDEX_NAME      LEAF_BLOCKS STATUS
--------------- ----------- --------
BOWIE_CODE_IDX         1936 VALID
```

Table Move (and Index Maintenance)

Before 12.2

In session one

```
SQL> insert into bowie values (100001, 42, 'DAVID BOWIE');

1 row created.
```

In session two

```
SQL> alter table bowie move;
alter table bowie move
            *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired

SQL> alter table bowie move online;
alter table bowie move online
*
ERROR at line 1:
ORA-01735: invalid ALTER TABLE option
```

Table Move (and Index Maintenance)

Before 12.2

In session one

```
SQL> commit;

Commit complete.
```

In session two

```
SQL> alter table bowie move;

Table altered.

SQL> select index_name, leaf_blocks, status from dba_indexes where table_name='BOWIE';

INDEX_NAME      LEAF_BLOCKS STATUS
--------------- ----------- --------
BOWIE_CODE_IDX         1936 UNUSABLE
```

New 12.2: Online Table Move (and Index Maint)

12.2

In session one

```
SQL> insert into bowie values (100001, 42, 'DAVID BOWIE');
1 row created.
```

In session two

```
SQL> alter table bowie move online;     => Hangs
```

In session one

```
SQL> commit;
Commit complete.
```

In session two

```
Table altered.
SQL> select index_name, leaf_blocks, status from dba_indexes where table_name='BOWIE';
INDEX_NAME_____ LEAF_BLOCKS STATUS__
BOWIE_CODE_IDX         1939 VALID
```

Online Table Move: Improve Performance

```
SQL> create table ziggy (id number, code number, date_created date, name varchar2(42));

Table created.

SQL> insert into ziggy select rownum, mod(rownum,100), sysdate - mod(rownum,1000), 'DAVID BOWIE'
     from dual connect by level <= 2000000;

2000000 rows created.

SQL> commit;

Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>user, tabname=>'ZIGGY', estimate_percent=>null,
method_opt=>'FOR ALL COLUMNS SIZE 1');

PL/SQL procedure successfully completed.

SQL> create index ziggy_code_i on ziggy(code);

Index created.

SQL> select index_name, clustering_factor, num_rows from user_indexes
     where index_name='ZIGGY_CODE_I';

INDEX_NAME           CLUSTERING_FACTOR   NUM_ROWS
-------------------- ----------------- ----------
ZIGGY_CODE_I                   923900    2000000
```

Online Table Move: Improve Performance

```
SQL> set arraysize 5000
SQL> select * from ziggy where code = 42;

20000 rows selected.

Execution Plan
----------------------------------------------------------------------------------------
| Id  | Operation                 | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |       | 20000 |   546K|   2750  (1)| 00:00:01 |
|*  1 |  TABLE ACCESS STORAGE FULL| ZIGGY | 20000 |   546K|   2750  (1)| 00:00:01 |
----------------------------------------------------------------------------------------

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

   1 - storage("CODE"=42)
       filter("CODE"=42)

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       9354  consistent gets
          0  physical reads
          0  redo size
     367165  bytes sent via SQL*Net to client
        533  bytes received via SQL*Net from client
          5  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
      20000  rows processed
```

Online Table Move: Improve Performance

```
SQL> alter table ziggy add clustering by linear order(code) without materialized zonemap;

Table altered.

SQL> alter table ziggy move online;

Table altered.

SQL> select index_name, clustering_factor, num_rows, status from user_indexes
     where index_name='ZIGGY_CODE_I';

INDEX_NAME      CLUSTERING_FACTOR   NUM_ROWS STATUS
--------------- ----------------- ---------- ------
ZIGGY_CODE_I                 9277    2000000 VALID
```

Clustering Factor has reduced to just 9277 from the previous 923900

Online Table Move: Improve Performance

```
SQL> select * from ziggy where code=42;

20000 rows selected.

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              | 20000 |   546K|   126   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| ZIGGY        | 20000 |   546K|   126   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | ZIGGY_CODE_I | 20000 |       |    33   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

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

   2 - access("CODE"=42)

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        136  consistent gets
          0  physical reads
          0  redo size
     670725  bytes sent via SQL*Net to client
        533  bytes received via SQL*Net from client
          5  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
      20000  rows processed
```

New in 12.2: Online Conversion to Partitioned Table

Rather than maybe rebuilding the entire table on a routine basis, partitioning allow us to only rebuild the portion of the table that actually changes.

To fully demonstrate new capabilities, let's create a couple of additional indexes:

```
SQL> create unique index ziggy_id_i on ziggy(id);

Index created.

SQL> create index ziggy_date_created_i on ziggy(date_created);

Index created.
```

Online Conversion to Partitioned Table (Update Indexes)

```
SQL> alter table ziggy
  2  modify partition by range (date_created)
  3        (partition p1 values less than (TO_DATE('01-JAN-2016', 'DD-MON-YYYY')),
  4         partition p2 values less than (TO_DATE('01-JAN-2017', 'DD-MON-YYYY')),
  5         partition p3 values less than (maxvalue)) online;

Table altered.

SQL> select table_name, status, partitioned from dba_tables where table_name='ZIGGY';

TABLE_NAME   STATUS   PAR
------------ -------- ---
ZIGGY        VALID    YES
```

Online Conversion to Partitioned Table (Update Indexes)

```
SQL> select index_name, status, partitioned, num_rows from dba_indexes where table_name='ZIGGY';

INDEX_NAME           STATUS   PAR   NUM_ROWS
-------------------- -------- --- ----------
ZIGGY_DATE_CREATED_I N/A      YES    2000000
ZIGGY_CODE_I         VALID    NO     2000000
ZIGGY_ID_I           VALID    NO     2000000

SQL> select index_name, partition_name, status, leaf_blocks from dba_ind_partitions
     where index_name like 'ZIGGY%';

INDEX_NAME           PARTITION_NAME  STATUS   LEAF_BLOCKS
-------------------- --------------- -------- -----------
ZIGGY_DATE_CREATED_I P1              USABLE           865
ZIGGY_DATE_CREATED_I P2              USABLE          1123
ZIGGY_DATE_CREATED_I P3              USABLE          1089

SQL> select index_name, partitioning_type, partition_count, locality from dba_part_indexes
     where table_name='ZIGGY';

INDEX_NAME           PARTITION PARTITION_COUNT LOCALI
-------------------- --------- --------------- ------
ZIGGY_DATE_CREATED_I RANGE                   3 LOCAL
```

Online Conversion to Partitioned Table (Update Indexes)

Alternatively, can also explicitly convert indexes to also be partitioned with UPDATE INDEXES clause

```
SQL> alter table ziggy
  2  modify partition by range (date_created)
  3        (partition p1 values less than (TO_DATE('01-JAN-2016', 'DD-MON-YYYY')),
  4         partition p2 values less than (TO_DATE('01-JAN-2017', 'DD-MON-YYYY')),
  5         partition p3 values less than (maxvalue)) online
  6  update indexes
  7      (ziggy_code_i local,
  8       ziggy_id_i global partition by range (id)
  9       (partition ip1 values less than (maxvalue)));

Table altered.

SQL> select table_name, status, partitioned from dba_tables where table_name='ZIGGY';

TABLE_NAME   STATUS   PAR
------------ -------- ---
ZIGGY        VALID    YES
```

Online Conversion to Partitioned Table (Update Indexes)

```
SQL> select index_name, status, partitioned from dba_indexes
     where table_name='ZIGGY';

INDEX_NAME           STATUS   PAR
-------------------- -------- ---
ZIGGY_CODE_I         N/A      YES
ZIGGY_ID_I           N/A      YES
ZIGGY_DATE_CREATED_I N/A      YES

SQL> select index_name, partitioning_type, partition_count, locality
     from dba_part_indexes where table_name='ZIGGY';

INDEX_NAME           PARTITION PARTITION_COUNT LOCALI
-------------------- --------- --------------- ------
ZIGGY_CODE_I         RANGE                   3 LOCAL
ZIGGY_ID_I           RANGE                   1 GLOBAL
ZIGGY_DATE_CREATED_I RANGE                   3 LOCAL
```

Online Table Partition Move: (Improve Performance)

```
SQL> select partition_name, num_rows, clustering_factor from dba_ind_partitions
     where index_name='ZIGGY_CODE_I';

PARTITION_NAME       NUM_ROWS CLUSTERING_FACTOR
-------------------- -------- -----------------
P1                     490000              2275
P2                     730000              3388
P3                     780000              3620

SQL> insert into ziggy select 2000000+rownum, mod(rownum,100), sysdate, 'DAVID BOWIE'
     from dual connect by level <= 500000;

500000 rows created.

SQL> commit;

Commit complete.

SQL> exec dbms_stats.gather_index_stats(ownname=>null, indname=>'ZIGGY_CODE_I');

PL/SQL procedure successfully completed.

SQL> select partition_name, num_rows, clustering_factor from dba_ind_partitions
     where index_name='ZIGGY_CODE_I';

PARTITION_NAME       NUM_ROWS CLUSTERING_FACTOR
-------------------- -------- -----------------
P1                     490000              2275
P2                     730000              3388
P3                    1280000            238505    ← Much worse after inserts
```

Online Table Partition Move: (Improve Performance)

```
SQL> alter table ziggy move partition p3 update indexes online;

Table altered.

SQL> select partition_name, num_rows, clustering_factor
     from dba_ind_partitions where index_name='ZIGGY_CODE_I';

PARTITION_NAME          NUM_ROWS CLUSTERING_FACTOR
-------------------- ---------- -----------------
P1                       490000              2275
P2                       730000              3388
P3                      1280000              5978
```

Partitioning can enable only a relatively small proportion of a table having to be reorganized to maintain optimal performance…

Online Modify Partitioned Table To Partitioned Table (New 18c)

```
SQL> create table pink_floyd (id number, status varchar2(6), name varchar2(42))
 indexing off
 partition by range (id)
(partition pf1 values less than (1000001),
 partition pf2 values less than (2000001) indexing off,
 partition pf3 values less than (maxvalue) indexing on);

Table created.

SQL> insert into pink_floyd select rownum, 'CLOSED', 'DAVID BOWIE' from dual connect by level <= 3000000;

3000000 rows created.

SQL> update pink_floyd set status = 'OPEN' where id > 2000000 and mod(id,10000)=0;

100 rows updated.

SQL> commit;

Commit complete.
```

Online Modify Partitioned Table To Partitioned Table (New 18c)

```
SQL> exec dbms_stats.gather_table_stats(ownname=>user, tabname=>'PINK_FLOYD', estimate_percent=>null,
method_opt=>'FOR ALL COLUMNS SIZE 1 FOR COLUMNS STATUS SIZE 5');

PL/SQL procedure successfully completed.

SQL> create index pink_floyd_status_i on pink_floyd(status);

Index created.

SQL> select index_name, num_rows, leaf_blocks, indexing from user_indexes where index_name = 'PINK_FLOYD_STATUS_I';

INDEX_NAME             NUM_ROWS LEAF_BLOCKS INDEXING
-------------------- ---------- ----------- --------
PINK_FLOYD_STATUS_I     3000000        9203 FULL

SQL> alter index pink_floyd_status_i rebuild indexing partial;

Index altered.

SQL> select index_name, num_rows, leaf_blocks, indexing from user_indexes where index_name = 'PINK_FLOYD_STATUS_I';

INDEX_NAME             NUM_ROWS LEAF_BLOCKS INDEXIN
-------------------- ---------- ----------- -------
PINK_FLOYD_STATUS_I     1000000        3068 PARTIAL
```

Online Modify Partitioned Table To Partitioned Table (New 18c)

```
SQL> select * from pink_floyd where status = 'OPEN' and id >= 2000001;

100 rows selected.

Execution Plan
----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                     |   100 |  2500 |     4   (0)| 00:00:01 |       |       |
|*  1 |  TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED| PINK_FLOYD          |   100 |  2500 |     4   (0)| 00:00:01 |     3 |     3 |
|*  2 |   INDEX RANGE SCAN                           | PINK_FLOYD_STATUS_I |    33 |       |     3   (0)| 00:00:01 |       |       |
----------------------------------------------------------------------------------------------------------------------------------

SQL> select * from pink_floyd where status = 'OPEN';

100 rows selected.

Execution Plan
----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                     |   100 |  2500 |  2491   (9)| 00:00:01 |       |       |
|   1 |  VIEW                                        | VW_TE_2             |    99 |  4059 |  2491   (9)| 00:00:01 |       |       |
|   2 |   UNION-ALL                                  |                     |       |       |            |          |       |       |
|*  3 |    TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED| PINK_FLOYD          |    33 |   825 |     4   (0)| 00:00:01 | ROWID | ROWID |
|*  4 |     INDEX RANGE SCAN                         | PINK_FLOYD_STATUS_I |   100 |       |     3   (0)| 00:00:01 |       |       |
|   5 |    PARTITION RANGE ITERATOR                  |                     |    66 |  1650 |  2487   (9)| 00:00:01 |     1 |     2 |
|*  6 |     TABLE ACCESS FULL                        | PINK_FLOYD          |    66 |  1650 |  2487   (9)| 00:00:01 |     1 |     2 |
----------------------------------------------------------------------------------------------------------------------------------
```

NEW IN 18c

Online Modify Partitioned Table To Partitioned Table (New 18c)

```
SQL> alter table pink_floyd
 modify
 partition by range (id) subpartition by list(status)
  subpartition template
 (subpartition closed values ('CLOSED') indexing off, subpartition open values ('OPEN') indexing on)
(partition pf1 values less than (1000001),
 partition pf2 values less than (2000001),
 partition pf3 values less than (maxvalue)) online;

Table altered.

SQL> select subpartition_position, subpartition_name, num_rows, indexing from user_tab_subpartitions where table_name =
'PINK_FLOYD';

SUBPARTITION_POSITION SUBPARTITION_NA   NUM_ROWS INDEXING
--------------------- --------------- ---------- --------
                    1 PF1_CLOSED         1000000 OFF
                    2 PF1_OPEN                 0 ON
                    1 PF2_CLOSED         1000000 OFF
                    2 PF2_OPEN                 0 ON
                    1 PF3_CLOSED          999900 OFF
                    2 PF3_OPEN                100 ON

SQL> select index_name, num_rows, leaf_blocks, indexing from user_indexes where index_name = 'PINK_FLOYD_STATUS_I';

INDEX_NAME             NUM_ROWS LEAF_BLOCKS INDEXIN
-------------------- ---------- ----------- -------
PINK_FLOYD_STATUS_I         100           1 PARTIAL
```

NEW IN 18c

Online Modify Partitioned Table To Partitioned Table (New 18c)

```
SQL> select * from pink_floyd where status = 'OPEN';

100 rows selected.

Execution Plan
----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                   | Name                | Rows | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                            |                     |  100 |  2500 |     2   (0)| 00:00:01 |       |       |
|   1 |  TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED| PINK_FLOYD          |  100 |  2500 |     2   (0)| 00:00:01 | ROWID | ROWID |
|*  2 |   INDEX RANGE SCAN                          | PINK_FLOYD_STATUS_I |  100 |       |     1   (0)| 00:00:01 |       |       |
----------------------------------------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          4  consistent gets
          0  physical reads
          0  redo size
       3315  bytes sent via SQL*Net to client
        608  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        100  rows processed
```

New in 12.2: Deferred Invalidation - Index Creation/Rebuild/Coalesce

```
SQL> create table bowie (id number, code number, name varchar2(42));

Table created.

SQL> insert into bowie select rownum, mod(rownum,10000), 'DAVID BOWIE'
     from dual connect by level <= 1000000;

1000000 rows created.

SQL> commit;

Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE');

PL/SQL procedure successfully completed.

SQL> create index bowie_id_i on bowie(id);

Index created.
```

Deferred Invalidation

```
SQL> select * from bowie where id=42;   ===> execute a couple of times

Execution Plan
----------------------------------------------------------------------------------------------
| Id  | Operation                            | Name       | Rows  | Bytes |  Cost(%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |            |     1 |    21 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE      |     1 |    21 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                   | BOWIE_ID_I |     1 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------

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

Deferred Invalidation

```
SQL> alter index bowie_id_i rebuild;
Index altered.
SQL> select * from bowie where id=42;

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Rows  | Bytes |  Cost(%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |     1 |    21 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE      |     1 |    21 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | BOWIE_ID_I |     1 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

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

Deferred Invalidation

```
SQL> alter index bowie_id_i rebuild deferred invalidation;
Index altered.
SQL> select * from bowie where id=42;

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Rows  | Bytes | Cost(%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |     1 |    21 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE      |     1 |    21 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | BOWIE_ID_I |     1 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

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

Deferred Invalidation

```
SQL> select * from bowie where code=42;  => Currently no index exists on the code column

100 rows selected.

------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |   100 |  2100 |  1023   (11)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| BOWIE |   100 |  2100 |  1023   (11)| 00:00:01 |
------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
       3603  consistent gets
          0  physical reads
          0  redo size
       2880  bytes sent via SQL*Net to client
        674  bytes received via SQL*Net from client
          8  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        100  rows processed
```

Deferred Invalidation

```
SQL> create index bowie_code_i on bowie(code);
Index created.
SQL> select * from bowie where id=42;
Execution Plan
----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Rows  | Bytes | Cost(%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |     1 |    21 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE      |     1 |    21 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | BOWIE_ID_I |     1 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------
Statistics
----------------------------------------------------------
          1  recursive calls    => Note has impacted validation of other queries on bowie table
          0  db block gets
          5  consistent gets
         12  physical reads
          0  redo size
        687  bytes sent via SQL*Net to client
        608  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Deferred Invalidation

```
SQL> select * from bowie where code=42;

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |   100 |  2100 |   103   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE        |   100 |  2100 |   103   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | BOWIE_CODE_I |   100 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          1  recursive calls   => As we would want, has invalidated query with code in predicates
          0  db block gets
        110  consistent gets
         16  physical reads
          0  redo size
       4376  bytes sent via SQL*Net to client
        674  bytes received via SQL*Net from client
          8  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        100  rows processed
```

Deferred Invalidation

```
SQL> drop index bowie_code_i deferred invalidation;

Index dropped.

SQL> select * from bowie where code=42;

Execution Plan
----------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |   100 |  2100 |  1023   (11)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| BOWIE |   100 |  2100 |  1023   (11)| 00:00:01 |
----------------------------------------------------------------------------

Statistics
----------------------------------------------------------
         46  recursive calls
          0  db block gets
       3640  consistent gets
          0  physical reads
          0  redo size
       2880  bytes sent via SQL*Net to client
        674  bytes received via SQL*Net from client
          8  SQL*Net roundtrips to/from client
          6  sorts (memory)
          0  sorts (disk)
        100  rows processed
```

Deferred Invalidation

```
SQL> create index bowie_code_i on bowie(code) deferred invalidation;

Index created.

SQL> select * from bowie where id=42;

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Rows  | Bytes | Cost(%CPU) | Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |     1 |    21 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE      |     1 |    21 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | BOWIE_ID_I |     1 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          0  recursive calls   => Has not impacted other queries based on Bowie table
          0  db block gets
          5  consistent gets
          0  physical reads
          0  redo size
        687  bytes sent via SQL*Net to client
        608  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Deferred Invalidation

```
SQL> select * from bowie where code=42;

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |   100 |  2100 |   103   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE        |   100 |  2100 |   103   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | BOWIE_CODE_I |   100 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       3603  consistent gets
          0  physical reads
          0  redo size
       2880  bytes sent via SQL*Net to client
        674  bytes received via SQL*Net from client
          8  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        100  rows processed
```

So consistent gets shows CBO still using a FTS (example of autotrace being deceptive)

Indexing Case In-Sensitive Database

```
SQL> create table bowie (id number, name varchar2(42), alias varchar2(42));

Table created.

SQL> insert into bowie select rownum, 'ZIGGY STARDUST'||rownum, 'ZIGGY'||rownum
     from dual connect by level <= 999995;

999995 rows created.

SQL> insert into bowie values (999996, 'David Bowie', 'Bowie');

1 row created.

SQL> insert into bowie values (999997, 'DAVID BOWIE', 'BOWIE');

1 row created.

SQL> insert into bowie values (999998, 'david bowie', 'bowie');

1 row created.

SQL> insert into bowie values (999999, 'David bowie', 'BowiE');

1 row created.

SQL> insert into bowie values (1000000, 'DaViD BoWiE', 'BoWiE');

1 row created.

SQL> commit;

Commit complete.
```

Indexing Case In-Sensitive Database

```
SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE');

PL/SQL procedure successfully completed.

SQL> create index bowie_name_i on bowie(name);

Index created.

SQL> select * from bowie where name='DAVID BOWIE';

        ID NAME            ALIAS
---------- --------------- ----------
    999997 DAVID BOWIE     BOWIE

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |     1 |    38 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE        |     1 |    38 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | BOWIE_NAME_I |     1 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------
```

Database is case-sensitive by default…

Indexing Case In-Sensitive Database

```
SQL> select * from bowie where upper(name)='DAVID BOWIE';

        ID NAME            ALIAS
---------- --------------- ----------
    999996 David Bowie     Bowie
    999997 DAVID BOWIE     BOWIE
    999998 david bowie     bowie
    999999 David bowie     BowiE
   1000000 DaViD BoWiE     BoWiE


Execution Plan
----------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       | 10000 |   371K|  1674   (10)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| BOWIE | 10000 |   371K|  1674   (10)| 00:00:01 |
----------------------------------------------------------------------------
```

UPPER (or any other) function negates use of normal index and requires application to be modified…

Linguistic Index

```
SQL> CREATE INDEX bowie_name_ci_i ON bowie(NLSSORT(name,'NLS_SORT=BINARY_CI'));

Index created.

SQL> ALTER SESSION SET NLS_SORT='BINARY_CI';

Session altered.

SQL> ALTER SESSION SET NLS_COMP='LINGUISTIC';

Session altered.

SQL> select * from bowie where name='DAVID BOWIE';

         ID NAME            ALIAS
---------- --------------- ----------
     999996 David Bowie     Bowie
     999997 DAVID BOWIE     BOWIE
     999998 david bowie     bowie
     999999 David bowie     BowiE
    1000000 DaViD BoWiE     BoWiE

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name            | Rows  | Bytes |Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                 | 10000 |   253K| 1160   (1)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| BOWIE           | 10000 |   253K| 1160   (1)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | BOWIE_NAME_CI_I |  4000 |       | 1883   (1)| 00:00:01 |
----------------------------------------------------------------------------------------------------
```

A Linguistic Index can provide a case-insensitive option, without re-writing application

Linguistic Index

```
SQL> create index bowie_alias_i on bowie(alias);

Index created.

SQL> select * from bowie where alias='BOWIE';

         ID NAME            ALIAS
---------- --------------- ----------
    999996 David Bowie     Bowie
    999997 DAVID BOWIE     BOWIE
    999998 david bowie     bowie
    999999 David bowie     BowiE
   1000000 DaViD BoWiE     BoWiE

Execution Plan
----------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |     1 |    38 |  1750   (14)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| BOWIE |     1 |    38 |  1750   (14)| 00:00:01 |
----------------------------------------------------------------------------

SQL> select index_name, index_type from dba_indexes where table_name='BOWIE';

INDEX_NAME                     INDEX_TYPE
------------------------------ ---------------------------
BOWIE_NAME_I                   NORMAL
BOWIE_NAME_CI_I                FUNCTION-BASED NORMAL
BOWIE_ALIAS_I                  NORMAL
```

Use of session/system parameters negates use of binary indexes and case-sensitive searches…

New in 12.2: Case In-Sensitive Database

```
SQL> create table ziggy (id number, name varchar2(42), alias varchar2(42) collate binary) default collation binary_ci;

Table created.

SQL> insert into ziggy select rownum, 'ZIGGY STARDUST'||rownum, 'ZIGGY'||rownum
     from dual connect by level <= 999995;

999995 rows created.

SQL> insert into ziggy values (999996, 'David Bowie', 'Bowie');

1 row created.

SQL> insert into ziggy values (999997, 'DAVID BOWIE', 'BOWIE');

1 row created.

SQL> insert into ziggy values (999998, 'david bowie', 'bowie');

1 row created.

SQL> insert into ziggy values (999999, 'David bowie', 'BowiE');

1 row created.

SQL> insert into ziggy values (1000000, 'DaViD BoWiE', 'BoWiE');

1 row created.

SQL> commit;

Commit complete.
```

Indexing Case In-Sensitive Database

```
SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'ZIGGY');

PL/SQL procedure successfully completed.

SQL> create index ziggy_name_i on bowie(name);

Index created.

SQL> select * from ziggy where name='DAVID BOWIE';

         ID NAME            ALIAS
---------- --------------- ----------
    999996 David Bowie     Bowie
    999997 DAVID BOWIE     BOWIE
    999998 david bowie     bowie
    999999 David bowie     BowiE
   1000000 DaViD BoWiE     BoWiE

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |     1 |    38 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| ZIGGY        |     1 |    38 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | ZIGGY_NAME_I |     1 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------
```

The NAME column is now automatically case in-sensitive with an implicit Linguistic Index…

Indexing Case In-Sensitive Database

```
SQL> create index ziggy_alias_i on ziggy(alias);

Index created.

SQL> select * from ziggy where alias='BOWIE';

        ID NAME            ALIAS
---------- --------------- ----------
    999997 DAVID BOWIE     BOWIE

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |     1 |    26 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| ZIGGY         |     1 |    26 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | ZIGGY_ALIAS_I |     1 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

SQL> select index_name, index_type from dba_indexes where table_name='ZIGGY';

INDEX_NAME                     INDEX_TYPE
------------------------------ ---------------------------
ZIGGY_NAME_I                   FUNCTION-BASED NORMAL
ZIGGY_ALIAS_I                  NORMAL
```

The ALIAS column however remains case sensitive with an implicit Binary index …

Indexing Case In-Sensitive Database

```
SQL> alter table ziggy default collation binary; =➔ Only impacts new columns

Table altered.

SQL> select * from ziggy where name='DAVID BOWIE';

        ID NAME            ALIAS
---------- --------------- ----------
    999996 David Bowie     Bowie
    999997 DAVID BOWIE     BOWIE
    999998 david bowie     bowie
    999999 David bowie     BowiE
   1000000 DaViD BoWiE     BoWiE

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |               |     5 |   210 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| ZIGGY         |     5 |   210 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                    | ZIGGY_NAME_I |     5 |       |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

SQL> alter user bowie default collation binary_ci;

User altered.

SQL> alter session set default_collation=BINARY_CI;

Session altered.
```

The default Collation can be changed at the session, database or schema levels

JSON in the Oracle 12c Database

```
SQL> CREATE TABLE ziggy_json
  2  (id          number,
  3   ziggy_date date,
  4   ziggy_order CLOB
  5  CONSTRAINT ziggy_json_check CHECK (ziggy_order IS JSON));

Table created.

SQL> insert into ziggy_json values (1, sysdate, '{"This is not legal JSON"}');
insert into ziggy_json values (1, sysdate, '{"This is not legal JSON"}')
*
ERROR at line 1:
ORA-02290: check constraint (BOWIE.ZIGGY_JSON_CHECK) violated
```

JSON in the Oracle 12c Database

```
SQL> insert into ziggy_json
  2  select
  3      rownum,
  4      SYSdate,
  5      '{"PONumber"             : ' || rownum || ',
  6          "Reference"            : "2017' || rownum || 'DBOWIE",
  7        "Requestor"            : "David Bowie",
  8        "User"                 : "DBOWIE",
  9        "CostCenter"           : "A42",
 10        "ShippingInstructions" : {"name"   : "David Bowie",
 11                                  "Address": {"street"  : "42 Ziggy Street",
 12                                              "city"    : "Canberra",
 13                                              "state"   : "ACT",
 14                                              "zipCode" : 2601,
 15                                              "country" : "Australia"},
 16                                  "Phone" : [{"type" : "Office", "number" : "417-555-7777"},
 17                                             {"type" : "Mobile", "number" : "417-555-1234"}]},
 18        "Special Instructions" : null,
 19        "AllowPartialShipment" : true,
 20        "LineItems"            : [{"ItemNumber" : 1,
 21                                   "Part"       : {"Description" : "Hunky Dory",
 22                                                   "UnitPrice"   : 10.95},
 23                                   "Quantity"   : 5.0},
 24                                  {"ItemNumber" : 2,
 25                                   "Part"       : {"Description" : "Pin-Ups",
 26                                                   "UnitPrice"   : 10.95},
 27                                   "Quantity"   : 3.0}]}'
 28  from dual connect by level <= 1000000;
1000000 rows created.
SQL> insert into ziggy_json select * from ziggy_json;
1000000 rows created.
SQL> commit;
Commit complete
```

JSON Text Index

```
SQL> CREATE INDEX ziggy_search_i ON ziggy_json (ziggy_order)
  2  INDEXTYPE IS CTXSYS.CONTEXT
  3  PARAMETERS ('section group CTXSYS.JSON_SECTION_GROUP SYNC (ON COMMIT)');

Index created.

SQL> SELECT * FROM ziggy_json WHERE json_textcontains(ziggy_order, '$.Reference', '201742DBOWIE');

Elapsed: 00:00:00.01

Execution Plan
----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |     2 |  5982 |     7   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ZIGGY_JSON     |     2 |  5982 |     7   (0)| 00:00:01 |
|*  2 |   DOMAIN INDEX               | ZIGGY_SEARCH_I |       |       |     4   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("CTXSYS"."CONTAINS"("ZIGGY_JSON"."ZIGGY_ORDER",'201742DBOWIE INPATH(/Reference)')>0)

Statistics
----------------------------------------------------------
         46  recursive calls
          0  db block gets
         63  consistent gets
          0  physical reads
          0  redo size
       2135  bytes sent via SQL*Net to client
       1200  bytes received via SQL*Net from client
          6  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

New In 12.2: JSON Text Search Index

```
SQL> drop index ziggy_search_i;

Index dropped.

SQL> CREATE SEARCH INDEX ziggy_search_idx ON ziggy_json (ziggy_order) FOR JSON;

Index created.
```

- Synchronous by default
- Both text and numeric ranges indexed by default
- Includes JSON Data Guide (structural information) by default
- Can be created on range and list partitioned tables

JSON Search Index: Attribute Searches

```
SQL> SELECT * FROM ziggy_json WHERE json_textcontains(ziggy_order, '$.Reference', '201742DBOWIE');

Elapsed: 00:00:00.00

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |     2 |  5982 |       7 (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ZIGGY_JSON       |     2 |  5982 |       7 (0)| 00:00:01 |
|*  2 |   DOMAIN INDEX               | ZIGGY_SEARCH_IDX |       |       |       4 (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

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

   2 - access("CTXSYS"."CONTAINS"("ZIGGY_JSON"."ZIGGY_ORDER",'201742DBOWIE INPATH(/Reference)')>0)

Statistics
----------------------------------------------------------
         51  recursive calls
          0  db block gets
         65  consistent gets
          0  physical reads
          0  redo size
       2135  bytes sent via SQL*Net to client
       1200  bytes received via SQL*Net from client
          6  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

JSON Search Index: Generic Searches

```
SQL> SELECT * FROM ziggy_json WHERE json_textcontains(ziggy_order, '$', '4242');

Elapsed: 00:00:00.00

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |     2 |  5982 |       7 (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ZIGGY_JSON       |     2 |  5982 |       7 (0)| 00:00:01 |
|*  2 |   DOMAIN INDEX               | ZIGGY_SEARCH_IDX |       |       |       4 (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

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

   2 - access("CTXSYS"."CONTAINS"("ZIGGY_JSON"."ZIGGY_ORDER",'4242')>0)

Statistics
----------------------------------------------------------
         19  recursive calls
          0  db block gets
         22  consistent gets
          0  physical reads
          0  redo size
       2137  bytes sent via SQL*Net to client
       1200  bytes received via SQL*Net from client
          6  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

JSON Search Index: Complex Searches

```
SQL> SELECT * FROM ziggy_json WHERE json_exists(ziggy_order, '$?(@.Reference =="2O1742DBOWIE" && @.LineItems.Quantity > 2)');

Elapsed: 00:00:00.01

Execution Plan
----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |     1 |  2991 |       4 (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| ZIGGY_JSON       |     1 |  2991 |       4 (0)| 00:00:01 |
|*  2 |   DOMAIN INDEX               | ZIGGY_SEARCH_IDX |       |       |       4 (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------

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

   1 - filter(JSON_EXISTS2("ZIGGY_ORDER" FORMAT JSON , '$?(@.Reference =="2O1742DBOWIE"&& @.LineItems.Quantity > 2)' FALSE ON ERROR)=1)
   2 - access("CTXSYS"."CONTAINS"("ZIGGY_JSON"."ZIGGY_ORDER",'{2O1742DBOWIE} INPATH(/Reference) and sdatap(CTXSYS.JSON_SEARCH_GROUPNUM*
       >   2 /LineItems/Quantity)')>0)

Statistics
----------------------------------------------------------
        143  recursive calls
          0  db block gets
        803  consistent gets
          0  physical reads
          0  redo size
       2135  bytes sent via SQL*Net to client
       1200  bytes received via SQL*Net from client
          6  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

255 Byte Key Length-JSON Search Indexes

Key length up from 64 bytes to 255 bytes; enables faster search queries for JSON objects containing long key names

```
SQL> create table bowie_json (id number, bowie_date date, bowie_order CLOB
     constraint bowie_json_check check (bowie_order IS JSON));

Table created.

SQL> insert into bowie_json
     select  rownum, sysdate,
     '{"PONumber"              : ' || rownum || ',
       "Reference"             : "2018' || rownum || 'DBOWIE",
       "Requestor"             : "David Bowie",
       "User"                  : "DBOWIE",
       "CostCenter"            : "A42",
"A_Really_Really__Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_
Really_Really_Really_Really_Really_Really_Really_Long_Description_Key" : "A description"}'
     from dual connect by level <= 1000;

1000 rows created.

SQL> commit;

Commit complete.
```

255 Byte Key Length-JSON Search Indexes

12.2 Example – Generated Error if key length greater than 64 bytes

```
SQL> create search index bowie_search_idx ON bowie_json (bowie_order) FOR JSON;

Index created.

SQL> select * from bowie_json where json_textcontains(bowie_order,
'$.A_Really_Really__Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Reall
y_Really_Really_Really_Really_Really_Really_Really_Long_Description_Key', 'David Bowie');
select * from bowie_json where json_textcontains(bowie_order,
'$.A_Really_Really__Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Reall
y_Really_Really_Really_Really_Really_Really_Really_Long_Description_Key', 'David Bowie')
              *
ERROR at line 1:
ORA-40467: JSON_TEXTCONTAINS() cannot be evaluated without a JSON-enabled
context index
```

NEW IN 18c

255 Byte Key Length-JSON Search Indexes

18c Example – Providing key is 255 byes or less, works

```
SQL> create search index bowie_search_idx ON bowie_json (bowie_order) FOR JSON;

Index created.

SQL> select * from bowie_json where json_textcontains(bowie_order,
'$.A_Really_Really__Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Really_Long_Description_Key', 'David Bowie');

no rows selected
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |     1 |  2036 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| BOWIE_JSON       |     1 |  2036 |     4   (0)| 00:00:01 |
|*  2 |   DOMAIN INDEX                | BOWIE_SEARCH_IDX |       |       |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   2 - access("CTXSYS"."CONTAINS"("BOWIE_JSON"."BOWIE_ORDER",'David Bowie INPATH
              (/A_Really_Really__Really_Really_Really_Really_Really_Really_Really_Really_Really_Really
              _Really_Really_Really_Really_Really_Really_Really_Really_Really_Long_Description_
              Key)')>0)
```

NEW IN 18c

Treat As JSON Function

Similar setup as before but this time the data is NOT defined as JSON with JSON check constraint

```
SQL> create table bowie (id number, bowie_date date, bowie_order CLOB);

Table created.

SQL> insert into bowie
     select  rownum, sysdate,
     '{"PONumber"    : ' || rownum || ',
       "Reference"   : "2018' || rownum || 'DBOWIE",
       "Requestor"   : "David Bowie",
       "User"        : "DBOWIE",
       "CostCenter"  : "A42",
       "Description" : "A description"}'
     from dual connect by level <= 1000;

1000 rows created.

SQL> commit;

Commit complete.
```

NEW IN 18c

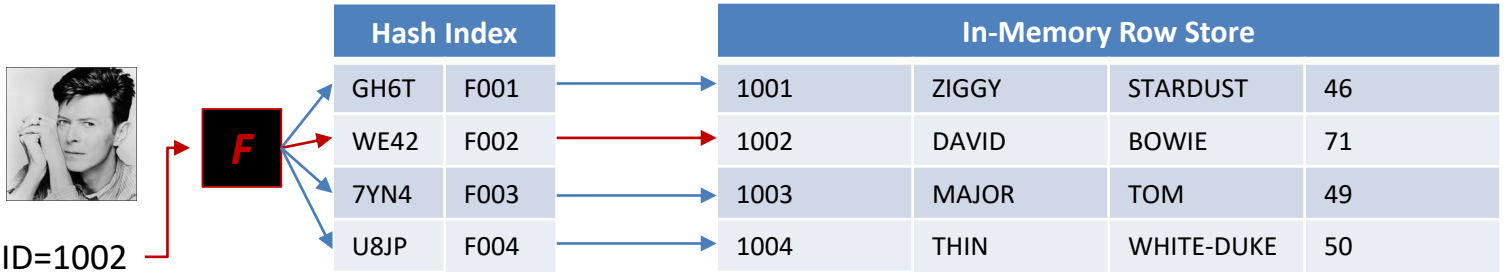
Treat As JSON Function

```
SQL> select b.bowie_order.PONumber, b.bowie_order.Reference from bowie b
     where b.bowie_order.PONumber='42';

ORA-00904: "B"."BOWIE_ORDER"."PONUMBER": invalid identifier

SQL> select b.bowie_json.PONumber, b.bowie_json.Reference
     from (select treat(bowie_order AS JSON) as bowie_json
           from bowie) b
     where b.bowie_json.PONumber='42';

PONUMBER REFERENCE
-------- ------------
42       201842DBOWIE
```

NEW IN 18c

Memoptimized Rowstore

- New SGA Memory Pool (new MEMOPTIMIZE_POOL_SIZE parameter)
- Uses a “lock free” In-Memory Hash Index
- 25% of the pool is automatically configured to store the In-Memory Hash Index
- Bypasses the SQL execution layer and executes directly in the data access layer
- Tables have a MEMOPTIMIZE FOR READ clause to assign in Memoptimized Pool
- Used only for equality Primary Key based conditions (ID=424242)
- Therefore table must have an enabled Primary Key
- DBMS_MEMOPTIMIZE.POPULATE procedure to populate tables in new pool



NEW IN 18c

Memoptimized Rowstore

```
SQL> create table bowie (id number, code number, name varchar2(42)) memoptimize for read;

ORA-62156: MEMOPTIMIZE FOR READ feature not allowed on segment with deferred storage

SQL> create table bowie (id number, code number, name varchar2(42)) segment creation
immediate memoptimize for read;

ORA-62142: MEMOPTIMIZE FOR READ feature requires NOT DEFERRABLE PRIMARY KEY constraint on the
table

SQL> create table bowie (id number constraint bowie_pk primary key, code number, name
varchar2(42)) segment creation immediate memoptimize for read;

Table created.
```

NEW IN 18c

Memoptimized Rowstore

```
SQL> select index_name, blevel, leaf_blocks from user_indexes where table_name='BOWIE';

INDEX_NAME BLEVEL LEAF_BLOCKS
---------- ------ -----------
BOWIE_PK        1         187

SQL> drop index bowie_pk;

ORA-02429: cannot drop index used for enforcement of unique/primary key
```

NEW IN 18c

Memoptimized Rowstore

```
SQL> explain plan for select * from bowie where id=42;

Statement processed.

SQL> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
Plan hash value: 1698080732

-----------------------------------------------------------------------------------------------
| Id  | Operation                                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |          |     1 |    20 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID READ OPTIM| BOWIE    |     1 |    20 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN READ OPTIM         | BOWIE_PK |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

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

   2 - access("ID"=42)
```

NEW IN 18c

Memoptimized Rowstore

```
explain plan for select * from bowie where id in (42, 442);

Statement processed.

SQL> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
Plan hash value: 3273446198

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |     2 |    40 |     3   (0)| 00:00:01 |
|   1 |  INLIST ITERATOR             |          |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| BOWIE    |     2 |    40 |     3   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN         | BOWIE_PK |     2 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   3 - access("ID"=42 OR "ID"=442)
```

NEW IN 18c

Memoptimized Rowstore

```
explain plan for select * from bowie where id=42 and name='DAVID BOWIE';

Statement processed.

SQL> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
Plan hash value: 1698080732

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |     1 |    21 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| BOWIE    |     1 |    21 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | BOWIE_PK |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   1 - filter("NAME"='DAVID BOWIE')
   2 - access("ID"=42)
```

Scalable Sequences

```
SQL> create sequence bowie_seq;

Sequence created.

SQL> select sequence_name, scale_flag, extend_flag from user_sequences
     where sequence_name='BOWIE_SEQ';

SEQUENCE_NAME   SCALE_FLAG EXTEND_FLAG
--------------- ---------- ------------
BOWIE_SEQ       N          N

SQL> select bowie_seq.nextval from dual;

   NEXTVAL
----------
         1

SQL> create table bowie (id number constraint bowie_id_i primary key, name varchar2(42));

Table created.
```

Scalable Sequences

```
SQL> create or replace procedure pop_bowie as
     begin
     for i in 1..100000 loop
     insert into bowie values (bowie_seq.nextval, 'DAVID BOWIE');
     commit;
     end loop;
     end;
     /

Procedure created.

SQL> exec pop_bowie   --- 3 sessions

PL/SQL procedure successfully completed.

SQL> analyze index bowie_id_i validate structure;

Index analyzed.

SQL> select name, lf_rows, lf_blks, pct_used from index_stats;

NAME          LF_ROWS    LF_BLKS   PCT_USED
---------- ---------- ---------- ----------
BOWIE_ID_I     300000        666         94
```

NEW IN 18c

Scalable Sequences

```
SQL> create sequence ziggy_seq scale;

Sequence created.

SQL> select sequence_name, scale_flag, extend_flag from user_sequences
     where sequence_name='ZIGGY_SEQ';

SEQUENCE_NAME    SCALE_FLAG EXTEND_FLAG
---------------- ---------- ------------
ZIGGY_SEQ        Y          N

SQL> select ziggy_seq.nextval from dual;

                            NEXTVAL
-----------------------------------
      101389000000000000000000001

SQL> create table ziggy (id number constraint ziggy_id_i primary key, name varchar2(42));

Table created.
```

6 digit prefix: 3 digits instance number and 3 digits session number

NEW IN 18c

Scalable Sequences

```
SQL> create or replace procedure pop_ziggy as
     begin
     for i in 1..100000 loop
     insert into ziggy values (ziggy_seq.nextval, 'DAVID BOWIE');
     commit;
     end loop;
     end;
     /

Procedure created.

SQL> exec pop_ziggy   --- 3 sessions

PL/SQL procedure successfully completed.

SQL> analyze index ziggy_id_i validate structure;

Index analyzed.

SQL> select name, lf_rows, lf_blks, pct_used from index_stats;

NAME          LF_ROWS    LF_BLKS   PCT_USED
---------- ---------- ---------- ----------
ZIGGY_ID_I     300000       1488         66   ➔ Previously 666 and 94 respectively
```

NEW IN 18c

Scalable Sequences

What's the problem ?

```
SQL> create sequence major_tom_seq maxvalue 9999 scale;

Sequence created.

SQL> select sequence_name, scale_flag, extend_flag from user_sequences
     where sequence_name='MAJOR_TOM_SEQ';

SEQUENCE_NAME  SCALE_FLAG EXTEND_FLAG
-------------- ---------- -----------
MAJOR_TOM_SEQ  Y          N

SQL> select major_tom_seq.nextval from dual;
select major_tom_seq.nextval from dual
       *
ERROR at line 1:
ORA-64603: NEXTVAL cannot be instantiated for MAJOR_TOM_SEQ. Widen the sequence
by 3 digits or alter sequence with SCALE EXTEND.
```

NEW IN 18c

Scalable Sequences

```
SQL> alter sequence major_tom_seq scale extend;

Sequence altered.

SQL> select sequence_name, scale_flag, extend_flag from user_sequences
     where sequence_name='MAJOR_TOM_SEQ';

SEQUENCE_NAME  SCALE_FLAG EXTEND_FLAG
-------------- ---------- -----------
MAJOR_TOM_SEQ  Y          Y

SQL> select major_tom_seq.nextval from dual;

   NEXTVAL
----------
1013890001
```

Automatic Indexing in Oracle Database 19c

Oracle OpenWorld 2018 Announcement !!

1. Capture
– Periodically capture the application SQL history into a SQL repository
– Includes SQL, plans, bind values, execution statistics, etc.
2. Identify Candidates
– Identify candidate indexes that may benefit the newly captured SQL statements
– Creates index candidates as unusable, invisible indexes (metadata only)
– Drop indexes obsoleted by newly created indexes (logical merge)
3. Verify
– Ask the optimizer if index candidates will be used for captured SQL statements
– Materialize indexes and run SQL to validate that the indexes improve their performance
– All verification is done outside application workflow

Automatic Indexing in Oracle Database 19c

4. Decide
– If performance is better for all statements, the indexes are marked visible
– If performance is worse for all statements, the indexes remain invisible
– If performance is worse for some, the indexes are marked visible except for the SQL statements that regressed

5. Online Validation
– The validation of the new indexes continues for other statements, online
– Only one of the sessions executing a SQL statement is allowed to use the new indexes

6. Monitor
– Index usage is continuously monitored
– Automatically created indexes that have not been used in a long time will be dropped

Supports
– Single and concatenated indexes
– Function-based indexes
– Compression Advanced Low

Oracle Indexing Internals and Best Practices 2 Day Seminar

Webinars Coming In March 2019 !!

Of benefit to DBAs, Developers, Solution Architects and anyone else interested in designing, developing or maintaining high performance Oracle-based applications/databases.

Indexes fundamental to every Oracle database and crucial for optimal performance. Many applications/databases are suboptimal and run inefficiently primarily because of inappropriate indexing strategies.

This highly acclaimed webinar covers many useful tips and strategies to maximise benefits of indexes on application/database performance and scalability, as well as in maximising Oracle database investments.

4-8 March & 26-30 March 2019 Limited Places Available !!

richardfooteconsulting.com/indexing-webinar

richardfoote.wordpress.com