Indexing New Features: Oracle 11g Release 1 and Release 2

Richard Foote
Richard Foote

- Working in IT for 24 years (scary stuff)
- Working with Oracle for 14 years (almost as scary)
- Previously employed by Oracle Corporation for 5½ years (scary as hell)
- Currently employed by the Australian Federal Police as a Senior DBA
- Responsible for many large scale, mission critical, “life-dependant” classified Oracle systems
- Based in sunny Canberra, Australia
- Oracle OakTable Member since 2002 and Oracle ACE Director since 2008
- Interests includes all sports and music (especially David Bowie, Pink Floyd and Radiohead)
- richard.foote@bigpond.com
So What Am I Going To Talk About ...

- Index Creation and Rebuild Locking Improvements
- Index Statistics and the Oracle 11g CBO
- Invisible Indexes
- Segment On Demand Indexes
- Zero Storage Unusable Indexes
- New Index Related Hints
- Analyze Validate Structure Improvements
- Bitmap-Join Indexes On IOT
- Virtual Columns Without Function-Based Indexes
Index Rebuild (and Creation) Locks

- Oracle’s had the INDEX REBUILD ONLINE option for a long while
- However, only available with Enterprise Edition
- Prevents Parallel Execution
- Still requires two table locks on the base table at the start and at the end of the indexing process
- These table locks can still causes locking issues (before 11g)
Index Rebuild Locks (Pre 11g)

Session 1

SQL> CREATE TABLE bowie_stuff AS SELECT rownum id, 'David Bowie' name
FROM dual CONNECT BY LEVEL <= 10000;
Table created.

SQL> CREATE INDEX bowie_stuff_i ON bowie_stuff(id);
Index created.

Session 2

SQL> INSERT INTO bowie_stuff VALUES (10001, 'Pink Floyd');
1 row created. (No commit or rollback)

Session 1

SQL> ALTER INDEX bowie_stuff_i REBUILD ONLINE;
Session hangs!! (Due to transaction in Session 2)
Index Rebuild Locks (Pre 11g)

Session 3

```sql
SQL> INSERT INTO bowie_stuff VALUES (10002, 'Radiohead');
Session hangs !! (Due to lock required by index rebuild in Session 1)
```

Session 2

```sql
SQL> COMMIT;
Commit complete.
```

Releases lock in session 3 and index rebuild is free to proceed but it will eventually get stuck again by uncommitted session 3 as it requires another lock to complete the rebuild process...

In session 2, perform another insert before session 3 commits ...

```sql
SQL> INSERT INTO bowie_stuff VALUES (10003, 'Iggy Pop');
Session hangs !! (Due to the second lock required by index rebuild in Session 1)
```
Index Rebuild Locks (Pre 11g)

Session 3

SQL> COMMIT;
Commit complete.

Session 1

Index altered.

Session 2

1 row created.
Index Rebuild Locks (11g)

Session 1

SQL> CREATE TABLE bowie_stuff AS SELECT rownum id, 'David Bowie' name FROM dual CONNECT BY LEVEL <= 10000;
Table created.

SQL> CREATE INDEX bowie_stuff_i ON bowie_stuff(id);
Index created.

Session 2

SQL> INSERT INTO bowie_stuff VALUES (10001, 'Pink Floyd');
1 row created. (No commit or rollback)

Session 1

SQL> ALTER INDEX bowie_stuff_i REBUILD ONLINE;
Session hangs !! (Due to transaction in Session 2)

Note: No difference at this point ...
Index Rebuild Locks (11g)

Session 3

```sql
SQL> INSERT INTO bowie_stuff VALUES (10002, 'Radiohead');
1 row created.
```

Big Difference !! The index table lock no longer locks out other transactions

Session 2

```sql
SQL> COMMIT;
Commit complete.
```

Releases lock in session 1 and index rebuild is free to proceed but will eventually get stuck by uncommitted session 3 as it requires another lock to complete the rebuild ...

In session 2, perform another insert before session 3 commits ...

```sql
SQL> INSERT INTO bowie_stuff VALUES (10003, 'Iggy Pop');
1 row created.
```

Again, not a problem as the second index rebuild lock impacts no other transactions
Index Rebuild (and Creation) Locks

• With 11g, far safer to create or rebuild an index during busy Production periods
• The index DDL might be locked out for periods of time and take a while to complete ...
• BUT it won’t lock out other transactions
• Of course, most index rebuilds are a waste of time but that’s another story
Bitmap-Join Indexes

• Useful index structure for Data Warehouses
• Can create a bitmap index on a table based on the column of another table
• Can potentially make joins unnecessary and associated SQL queries more efficient
• But has a big restriction prior to Oracle11g ...
**Bitmap-Join Indexes**

```sql
SQL> CREATE TABLE big_dwh_table (id NUMBER PRIMARY KEY, album_id NUMBER, artist_id NUMBER, country_id NUMBER, format_id NUMBER, release_date DATE, total_sales NUMBER);
Table created.

SQL> CREATE SEQUENCE dwh_seq;
Sequence created.

SQL> create or replace procedure pop_big_dwh_table as
  2  v_id number;
  3  v_artist_id number;
  4  begin
  5    for v_album_id in 1..10000 loop
  6      v_artist_id:= ceil(dbms_random.value(0,100));
  7      for v_country_id in 1..100 loop
  8        select dwh_seq.nextval into v_id from dual;
  9        insert into big_dwh_table values (v_id, v_album_id, v_artist_id, v_country_id, ceil(dbms_random.value(0,4)), trunc(sysdate-mod(v_id,ceil(dbms_random.value(0,1000)))), ceil(dbms_random.value(0,500000)));
 10      end loop;
 11    end loop;
 12    commit;
 13 end;
 14 /
Procedure created.

SQL> exec pop_big_dwh_table
PL/SQL procedure successfully completed.

SQL> create bitmap index big_dwh_table_album_id_i on big_dwh_table(album_id);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>'BOWIE', tabname=>'BIG_DWH_TABLE', estimate_percent=>null, cascade=>true, method_opt=>'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
```
### Bitmap-Join Indexes

```sql
SQL> CREATE TABLE albums (album_id number, album_details varchar2(30));
Table created.

SQL> INSERT INTO albums SELECT rownum, substr(object_name,1,30) FROM dba_objects WHERE rownum <= 10000;
10000 rows created.

SQL> commit;
Commit complete.

SQL> alter table albums add primary key(album_id);
Table altered.

SQL> create index albums_details_i on albums(album_details);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=> 'Bowie', tabname=> 'ALBUMS', estimate_percent=> null, cascade=> true, method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
```
Bitmap-Join Indexes

SQL> SELECT b.id, b.album_id, b.format_id FROM big_dwh_table b, albums a WHERE b.album_id = a.album_id and a.album_details = 'TAB$';
100 rows selected.

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>125</td>
<td>4250</td>
<td>25 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
<td>125</td>
<td>4250</td>
<td>25 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ALBUMS</td>
<td>1</td>
<td>22</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>INDEX RANGE SCAN</td>
<td>ALBUMSDETAILS_I</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>BITMAP CONVERSION TO ROWIDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>BITMAP INDEX SINGLE VALUE</td>
<td>BIG_DWH_TABLE_ALBUM_ID_I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_DWH_TABLE</td>
<td>100</td>
<td>1200</td>
<td>25 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
10 consistent gets
0 physical reads
0 redo size
1648 bytes sent via SQL*Net to client
396 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
100 rows processed
Bitmap-Join Indexes

Let’s now create a Bitmap-Join Index ...

SQL> drop index albums_details_i;

Index dropped.

SQL> CREATE BITMAP INDEX big_dwh_album_details_i ON big_dwh_table(a.album_details)
    FROM big_dwh_table b, albums a
    WHERE b.album_id = a.album_id;

Index created.
Bitmap-Join Indexes

SQL> SELECT b.id, b.album_id, b.format_id FROM big_dwh_table b, albums a WHERE b.album_id = a.album_id and a.album_details = 'TAB$';

100 rows selected.

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>125</td>
<td>1500</td>
<td>26 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_DWH_TABLE</td>
<td>125</td>
<td>1500</td>
<td>26 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>BITMAP CONVERSION TO ROWIDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*  3</td>
<td>BITMAP INDEX SINGLE VALUE</td>
<td>BIG_DWH_ALBUM_DETAILS_I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics

0  recursive calls
0  db block gets
6  consistent gets
0  physical reads
0  redo size
1648 bytes sent via SQL*Net to client
396 bytes received via SQL*Net from client
2  SQL*Net roundtrips to/from client
0  sorts (memory)
0  sorts (disk)
100 rows processed

There is no join step in the execution plan, the ALBUMS table is not even referenced and consistent gets has dropped down from 10 to 6
SQL> drop table albums;
Table dropped.

SQL> CREATE TABLE albums (album_id number primary key, album_details varchar2(30)) organization index;
Table created.

SQL> INSERT INTO albums SELECT rownum, substr(object_name,1,30) FROM dba_objects WHERE rownum <= 10000;
10000 rows created.

SQL> commit;
Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>'BOWIE', tabname=>'ALBUMS', estimate_percent=>null, cascade=>true,
method_opt=>'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.

SQL> CREATE BITMAP INDEX big_dwh_album_details_i ON big_dwh_table(a.album_details)
FROM big_dwh_table b, albums a
WHERE b.album_id = a.album_id;
CREATE BITMAP INDEX big_dwh_album_details_i ON big_dwh_table(a.album_details)
*  
ERROR at line 1:
ORA-25966: join index cannot be based on an index organized table

However, can’t create a Bitmap-Join Index if either table is an Index Organized Table
11g Bitmap-Join Indexes on IOTs

However, since Oracle 11g Rel 1, Bitmap-Join Indexes on IOTs are fully supported ...

```sql
CREATE BITMAP INDEX big_dwh_album_details_i ON big_dwh_table(a.album_details) FROM big_dwh_table b, albums a WHERE b.album_id = a.album_id;
Index created.

SELECT b.id, b.album_id, b.format_id FROM big_dwh_table b, albums a WHERE b.album_id = a.album_id and a.album_details = 'TAB$'; 100 rows selected.
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>125</td>
<td>1500</td>
<td>26 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_DWH_TABLE</td>
<td>125</td>
<td>1500</td>
<td>26 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>BITMAP CONVERSION TO ROWIDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*3</td>
<td>BITMAP INDEX SINGLE VALUE</td>
<td>BIG_DWH_ALBUM_DETAILS_I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics

<table>
<thead>
<tr>
<th>recursive calls</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>db block gets</td>
<td>0</td>
</tr>
<tr>
<td>consistent gets</td>
<td>6</td>
</tr>
<tr>
<td>physical reads</td>
<td>0</td>
</tr>
<tr>
<td>redo size</td>
<td>0</td>
</tr>
<tr>
<td>bytes sent</td>
<td>1648</td>
</tr>
<tr>
<td>bytes received</td>
<td>396</td>
</tr>
<tr>
<td>SQL*Net roundtrips to/from client</td>
<td>2</td>
</tr>
<tr>
<td>sorts (memory)</td>
<td>0</td>
</tr>
<tr>
<td>sorts (disk)</td>
<td>0</td>
</tr>
<tr>
<td>rows processed</td>
<td>100</td>
</tr>
</tbody>
</table>
Invisible Indexes
SQL> SELECT * FROM invisible_bowie
    WHERE date_field BETWEEN '25-DEC-2006' AND '26-DEC-2006';

100 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost(%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>300</td>
<td>3600</td>
<td>304 (0)</td>
<td>00:00:04</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>INVISIBLE_BOWIE</td>
<td>300</td>
<td>3600</td>
<td>304 (0)</td>
<td>00:00:04</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>INVISIBLE_BOWIE_I</td>
<td>300</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Note: By default, indexes are “visible” to the CBO and can potentially be used
Invisible Indexes

If an index:
- becomes problematic, or
- you think index is not being used and might be safe to drop

you can make it “invisible” ...

```sql
SQL> ALTER INDEX invisible_bowie_i INVISIBLE;
Index altered.

SQL> SELECT index_name, visibility FROM user_indexes WHERE index_name = 'INVISIBLE_BOWIE_I';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>VISIBILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVISIBLE_BOWIE_I</td>
<td>INVISIBLE</td>
</tr>
</tbody>
</table>
```
Invisible Indexes

```
SQL> SELECT * FROM invisible_bowie
      WHERE date_field BETWEEN '25-DEC-2006' AND '26-DEC-2006';
100 rows selected.
```

Execution Plan

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>300</td>
<td>3600</td>
<td>699</td>
<td>00:00:09</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>INVISIBLE_BOWIE</td>
<td>300</td>
<td>3600</td>
<td>699</td>
<td>00:00:09</td>
</tr>
</tbody>
</table>
```

Note: The index is now invisible and is not considered by the CBO ...
Invisible Indexes

```
SQL> ALTER SESSION SET OPTIMIZER_USE_INVISIBLE_INDEXES = true;
Session altered.

SQL> SELECT * FROM invisible_bowie WHERE date_field > sysdate - 1;
100 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost(%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>100</td>
<td>1200</td>
<td>103 (0)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>INVISIBLE_BOWIE</td>
<td>100</td>
<td>1200</td>
<td>103 (0)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>INVISIBLE_BOWIE_I</td>
<td>100</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
```

Note: You can alter a session to make invisible indexes visible to the session ...

©Richard Foote - Indexing New Features: Oracle 11g Release 1 and Release 2
Invisible Indexes

SQL> ALTER INDEX invisible_bowie_i VISIBLE;
Index altered.

SQL> ALTER SESSION SET OPTIMIZER_USE_INVISIBLE_INDEXES = false;
Session altered.

SQL> SELECT * FROM invisible_bowie WHERE date_field > sysdate - 1;
100 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>100</td>
<td>1200</td>
<td>103 (0)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>INVISIBLE_BOWIE</td>
<td>100</td>
<td>1200</td>
<td>103 (0)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>INVISIBLE_BOWIE_I</td>
<td>100</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Note: An index can instantly be converted visible again as the index is always being maintained by Oracle, even when invisible ...
Invisible Indexes – Foreign Key Indexes

First, create a “Parent” table with 3 parent values ...

```
SQL> create table daddy (id number constraint daddy_pk primary key, name varchar2(20));
Table created.

SQL> insert into daddy values (1, 'BOWIE');
1 row created.

SQL> insert into daddy values (2, 'ZIGGY');
1 row created.

SQL> insert into daddy values (3, 'THIN WHITE DUKE');
1 row created.

SQL> commit;
Commit complete.
```
Invisible Indexes – Foreign Key Indexes

Next create a “Child” table that references the table with an **indexed** FK constraint

```sql
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', estimate_percent=>null, cascade=>true,
method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'KIDDIE', estimate_percent=>null, cascade=>true,
method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
```
Invisible Indexes – Foreign Key Indexes

SQL> delete daddy where id = 2;

1 row deleted.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DELETE STATEMENT</td>
<td></td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>DELETE</td>
<td>DADDY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C009714</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

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

At 8 db block gets, Oracle has used index on FK to determine whether any child rows exist ...
Invisible Indexes – Foreign Key Indexes

In Oracle 11g **Release 1**

```sql
SQL> rollback;
Rollback complete.

SQL> alter index kiddie_fk_i invisible;
Index altered.
```

If now make the index Invisible, does this change the manner in which the delete operation is performed?
Invisible Indexes – Foreign Key Indexes

SQL> delete daddy where id = 2;

1 row deleted.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DELETE STATEMENT</td>
<td></td>
<td>1</td>
<td>3</td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>DELETE</td>
<td>DADDY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C009714</td>
<td>1</td>
<td>3</td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

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

No !! The index, although invisible is still being used ...
Invisible Indexes—Foreign Key Indexes

SQL> drop index kiddie_fk_i;
Index dropped.

SQL> delete daddy where id = 3;
1 row deleted.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DELETE STATEMENT</td>
<td></td>
<td>1</td>
<td>3</td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>DELETE</td>
<td>DADDY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*  2</td>
<td>INDEX UNIQUE SCAN</td>
<td>SYS_C009714</td>
<td>1</td>
<td>3</td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

- 7 recursive calls
- 7 db block gets
- **3172** consistent gets
- 0 physical reads
- 632 redo size
- 674 bytes sent via SQL*Net to client
- 554 bytes received via SQL*Net from client
- 3 SQL*Net roundtrips to/from client
- 1 sorts (memory)
- 0 sorts (disk)
- 1 rows processed

Note: dropped index can’t be used to lookup child table and FTS on “KIDDIE” table is performed
Index Monitoring – Foreign Key Trap

```
SQL> select * from v$object_usage where index_name = 'KIDDIE_FK_I';

INDEX_NAME    TABLE_NAME  MON  USE  START_MONITORING     END_MONITORING
-----------    --------    ---  ---  -------------------     -------------------
KIDDIE_FK_I    KIDDIE     YES  NO   09/10/2008 12:10:52

SQL> drop index kiddie_fk_i;

Index dropped.
```

Note: Index Monitoring (in 11g Rel 1 and 2) also does not picked up the fact the index on the FK column of the KIDDIE table was actually used to ensure no record has the deleted value (2) ...

Deleting such a so-called “unused” index can be disastrous !!
### Pre 11g example (Note: There are only 10 distinct combinations of data or 10% selectivity)

```sql
SQL> create table radiohead (id number, code varchar2(5), name varchar2(20));
Table created.

SQL> begin
  2  for i in 1..10000 loop
  3    insert into radiohead values(1, 'AAA', 'Description A');
  4    insert into radiohead values(2, 'BBB', 'Description B');
  5    insert into radiohead values(3, 'CCC', 'Description C');
  6    insert into radiohead values(4, 'DDD', 'Description D');
  7    insert into radiohead values(5, 'EEE', 'Description E');
  8    insert into radiohead values(6, 'FFF', 'Description F');
  9    insert into radiohead values(7, 'GGG', 'Description G');
 10    insert into radiohead values(8, 'HHH', 'Description H');
 11    insert into radiohead values(9, 'III', 'Description I');
 12    insert into radiohead values(10, 'JJJ', 'Description J');
 13  end loop;
 14  commit;
 15  end;
 16  /
PL/SQL procedure successfully completed.

SQL> create index radiohead_idx on radiohead(id, code);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'RADIOHEAD', estimate_percent=>null, cascade=>true, method_opt=>'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
```
Index Statistics and the 11g CBO

SQL> select * from radiohead where id = 2 and code = 'BBB';

10000 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1000</td>
<td>21000</td>
<td>42 (10)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>RADIOHEAD</td>
<td>1000</td>
<td>21000</td>
<td>42 (10)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

1 recursive calls
0 db block gets
365 consistent gets
0 physical reads
0 redo size
50684 bytes sent via SQL*Net to client
407 bytes received via SQL*Net from client
3 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
10000 rows processed

Note: Oracle assumes the selectivity will be 10 x 10 = 100 distinct values (1%), not 10% because there are 10 distinct values of both the ID and CODE columns...
11g Rel 1 example (Note: There are only 10 distinct combinations of data or 10% selectivity)

```
SQL> create table radiohead (id number, code varchar2(5), name varchar2(20));
Table created.

SQL> begin
  2  for i in 1..10000 loop
  3    insert into radiohead values(1, 'AAA', 'Description A');
  4    insert into radiohead values(2, 'BBB', 'Description B');
  5    insert into radiohead values(3, 'CCC', 'Description C');
  6    insert into radiohead values(4, 'DDD', 'Description D');
  7    insert into radiohead values(5, 'EEE', 'Description E');
  8    insert into radiohead values(6, 'FFF', 'Description F');
  9    insert into radiohead values(7, 'GGG', 'Description G');
 10    insert into radiohead values(8, 'HHH', 'Description H');
 11    insert into radiohead values(9, 'III', 'Description I');
 12    insert into radiohead values(10, 'JJJ', 'Description J');
 13  end loop;
 14  commit;
 15  end;
 16  /
PL/SQL procedure successfully completed.

SQL> create index radiohead_idx on radiohead(id, code);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'RADIOHEAD', estimate_percent=>null, cascade=>true,
method_opt=>'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
```
Index Statistics and the 11g CBO

SQL> select * from radiohead where id = 2 and code = 'BBB';

10000 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10000</td>
<td>205K</td>
<td>100 (1)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>RADIOHEAD</td>
<td>10000</td>
<td>205K</td>
<td>100 (1)</td>
<td>00:00:02</td>
</tr>
</tbody>
</table>

Statistics

181  recursive calls
0    db block gets
391  consistent gets
0    physical reads
0    redo size
50684 bytes sent via SQL*Net to client
407  bytes received via SQL*Net from client
3    SQL*Net roundtrips to/from client
6    sorts (memory)
0    sorts (disk)
10000 rows processed

Note: In 11g, Oracle gets the cardinality estimate exactly correct !!
Note: In 11g, Oracle can use the index DISTINCT_KEYS statistic to determine the selectivity of a predicate

There are only 10 distinct index key values, therefore one combination of values will return 10% of the data ...
After dropping the index, the CBO now gets cardinality estimates incorrect ...
Index Statistics and the 11g CBO

Let’s create another table ...

```sql
SQL> create table ok_computer (id number, code varchar2(5), description varchar2(20));
Table created.

SQL> begin
  2  insert into ok_computer values(1, 'AAA', 'Description A');
  3  insert into ok_computer values(2, 'BBB', 'Description B');
  4  insert into ok_computer values(3, 'CCC', 'Description C');
  5  insert into ok_computer values(4, 'DDD', 'Description D');
  6  insert into ok_computer values(5, 'EEE', 'Description E');
  7  insert into ok_computer values(6, 'FFF', 'Description F');
  8  insert into ok_computer values(7, 'GGG', 'Description G');
  9  insert into ok_computer values(8, 'HHH', 'Description H');
 10  insert into ok_computer values(9, 'III', 'Description I');
 11  insert into ok_computer values(10, 'JJJ', 'Description J');
 12  commit;
13  end;
14  /
PL/SQL procedure successfully completed.

SQL> alter table ok_computer add primary key(id, code);
Table altered.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'OK_COMPUTER', estimate_percent=>null, cascade=>true, method_opt=>'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
```
Index Statistics and the 11g CBO

SQL> CREATE INDEX radiohead_idx on radiohead(id, code);
Index created.

SQL> select * from radiohead r, ok_computer o where r.id = 5 and r.code = 'EEE';
100000 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>100K</td>
<td>4101K</td>
<td>989 (2)</td>
<td>00:00:12</td>
</tr>
<tr>
<td>1</td>
<td>MERGE JOIN CARTESIAN</td>
<td></td>
<td>100K</td>
<td>4101K</td>
<td>989 (2)</td>
<td>00:00:12</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>OK_COMPUTER</td>
<td>10</td>
<td>210</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>3</td>
<td>BUFFER SORT</td>
<td></td>
<td>10000</td>
<td>205K</td>
<td>987 (2)</td>
<td>00:00:12</td>
</tr>
<tr>
<td>* 4</td>
<td>TABLE ACCESS FULL</td>
<td>RADIOHEAD</td>
<td>10000</td>
<td>205K</td>
<td>99 (2)</td>
<td>00:00:02</td>
</tr>
</tbody>
</table>

Statistics

1 recursive calls
0 db block gets
376 consistent gets
0 physical reads
0 redo size
1203277 bytes sent via SQL*Net to client
605 bytes received via SQL*Net from client
21 SQL*Net roundtrips to/from client
1 sorts (memory)
0 sorts (disk)
100000 rows processed

Note: although index not used in execution plan, it has used the index statistics to get the cardinality correct and uses a plan with 376 consistent gets.
Index Statistics and the 11g CBO

SQL> DROP INDEX radiohead_idx;
Index dropped.

SQL> select * from radiohead r, ok_computer o where r.id = 5 and r.code = 'EEE';
100000 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>10000</td>
<td>410K</td>
<td>373 (1)</td>
<td>00:00:05</td>
</tr>
<tr>
<td>1</td>
<td>MERGE JOIN CARTESIAN</td>
<td>RADIOHEAD</td>
<td>10000</td>
<td>410K</td>
<td>373 (1)</td>
<td>00:00:05</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>RADIOHEAD</td>
<td>1000</td>
<td>21000</td>
<td>100 (1)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>3</td>
<td>BUFFER SORT</td>
<td></td>
<td>10</td>
<td>210</td>
<td>272 (0)</td>
<td>00:00:04</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>OK_COMPUTER</td>
<td>10</td>
<td>210</td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

- 165 recursive calls
- 0 db block gets
- 404 consistent gets
- 0 physical reads
- 0 redo size
- 2603137 bytes sent via SQL*Net to client
- 605 bytes received via SQL*Net from client
- 21 SQL*Net roundtrips to/from client
- 6 sorts (memory)
- 0 sorts (disk)
- 100000 rows processed

Note: Drop the index, Oracle now gets the cardinality wrong as it can’t use the index statistics and chooses a more expensive execution plan ...
 Index Statistics and the 11g CBO

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'RADIOHEAD', estimate_percent=>null, cascade=>true, method_opt=>'FOR COLUMNS (id, code) SIZE 1');
PL/SQL procedure successfully completed.

SQL> select * from radiohead r, ok_computer o where r.id = 5 and r.code = 'EEE';
100000 rows selected.

Execution Plan
------------------------------------------------------------------------------------
| Id  | Operation            | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------
| 0   | SELECT STATEMENT     |             | 100K  | 4101K | 989    (2)| 00:00:12 |
| 1   | MERGE JOIN CARTESIAN|             | 100K  | 4101K | 989    (2)| 00:00:12 |
| 2   | TABLE ACCESS FULL   | OK_COMPUTER | 10    | 210   | 2      (0)| 00:00:01 |
| 3   | BUFFER SORT         |             | 10000 | 205K  | 987    (2)| 00:00:12 |
|* 4  | TABLE ACCESS FULL   | RADIOHEAD   | 10000 | 205K  | 99     (2)| 00:00:02 |
------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
8  recursive calls
0  db block gets
378  consistent gets
0  physical reads
0  redo size
1203277 bytes sent via SQL*Net to client
605  bytes received via SQL*Net from client
21  SQL*Net roundtrips to/from client
1  sorts (memory)
0  sorts (disk)
100000 rows processed

Note: 11g extended statistics can also be used to collect accurate statistics across column combinations and restore the efficient execution plan ...
SQL> select * from radiohead where id = 2 and code = 'BBB';

10000 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>10000</td>
<td>205K</td>
<td>100 (1)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>RADIOHEAD</td>
<td>10000</td>
<td>205K</td>
<td>100 (1)</td>
<td>00:00:02</td>
</tr>
</tbody>
</table>

Statistics

- 181 recursive calls
- 0 db block gets
- 391 consistent gets
- 0 physical reads
- 0 redo size
- 50684 bytes sent via SQL*Net to client
- 407 bytes received via SQL*Net from client
- 3 SQL*Net roundtrips to/from client
- 6 sorts (memory)
- 0 sorts (disk)
- 10000 rows processed
Invisible Indexes: Index Statistics

SQL> alter index radiohead_idx invisible;
Index altered.

SQL> select * from radiohead where id = 2 and code = 'BBB';
10000 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>10000</td>
<td>205K</td>
<td>100 (1)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>RADIOHEAD</td>
<td>10000</td>
<td>205K</td>
<td>100 (1)</td>
<td>00:00:02</td>
</tr>
</tbody>
</table>

Statistics

181  recursive calls
0    db block gets
391  consistent gets
0    physical reads
0    redo size
50684 bytes sent via SQL*Net to client
407  bytes received via SQL*Net from client
3    SQL*Net roundtrips to/from client
6    sorts (memory)
0    sorts (disk)
10000 rows processed

In 11g Rel 1, Oracle stills sees index statistics of invisible indexes ...
“Visible” Invisible Indexes

• In Oracle11g Release 1, Invisible Indexes are still visible:
  – When used to check and police FK consistency
  – When use by the CBO to lookup index statistics

• Both these issues have been fixed in Oracle11g Release 2
Invisible Indexes: PK and UK Constraints

Unique Indexes and indexes used to police Primary Key and Unique Key constraints can also be made “Invisible”.

```sql
SQL> create table bowie (id number constraint bowie_pk primary key using index(create unique index bowie_pk_i on bowie(id)), name varchar2(20));
Table created.

SQL> insert into bowie select rownum, 'DAVID BOWIE' from dual connect by level <= 10000;
10000 rows created.

SQL> commit;
Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE', estimate_percent=>null, cascade=>true, method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
```
Invisible Indexes: PK and UK Constraints

SQL> select * from bowie where id = 42;

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>15</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BOWIE</td>
<td>1</td>
<td>15</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*  2</td>
<td>INDEX UNIQUE SCAN</td>
<td>BOWIE_PK_I</td>
<td>1</td>
<td></td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
3 consistent gets
0 physical reads
0 redo size
471 bytes sent via SQL*Net to client
396 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed
Invisible Indexes: PK and UK Constraints

SQL> alter index bowie_pk_i invisible;
       Index altered.

SQL> select * from bowie where id = 42;

    ID  NAME
----------  ---------------
     42  DAVID BOWIE

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>15</td>
<td>8 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>BOWIE</td>
<td>1</td>
<td>15</td>
<td>8 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics
-----------
0 recursive calls
0 db block gets
33 consistent gets
0 physical reads
0 redo size
471 bytes sent via SQL*Net to client
396 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed
Invisible Indexes: PK and UK Constraints

However, if you attempt to insert a duplicate row that violates the constraint ...

```sql
SQL> insert into bowie values (1, 'ZIGGY');
insert into bowie values (1, 'ZIGGY')
*  
ERROR at line 1:
ORA-00001: unique constraint (BOWIE.BOWIE_PK) violated
```

In both Oracle11g Release 1 & 2, the index is still “visible” in that it’s used to police the constraint.
Invisible Indexes: PK and UK Constraints

Same scenario if you use a Non-Unique Index to police a PK or UK constraint

SQL> create table bowie (id number constraint bowie_pk primary key using index(create index bowie_pk_i on bowie(id)), name varchar2(20));
Table created.

SQL> insert into bowie select rownum, 'DAVID BOWIE' from dual connect by level <= 10000;
10000 rows created.

SQL> commit;
Commit complete.

SQL> alter index bowie_pk_i invisible;
Index altered.

SQL> insert into bowie values (1, 'ZIGGY');
insert into bowie values (1, 'ZIGGY')
*
ERROR at line 1:
ORA-00001: unique constraint (BOWIE.BOWIE_PK) violated
Invisible Indexes: Unique Indexes

Same scenario if you just create a Unique Index ...

```
SQL> create table bowie as select rownum id, 'DAVID_BOWIE' name from dual connect by level <=10000;
Table created.

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

SQL> alter index bowie_id_i invisible;
Index altered.

SQL> insert into bowie values (1, 'ZIGGY');
insert into bowie values (1, 'ZIGGY')
* ERROR at line 1:
ORA-00001: unique constraint (BOWIE.BOWIE_ID_I) violated
```
Invisible Indexes: 11g Rel. 1 Bug

Nasty bug with Oracle11g Release 1 when it comes to collecting statistics on an Invisible index:

```
SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE', estimate_percent=>null, cascade=>true, method_opt=> 'FOR ALL COLUMNS SIZE 1');
BEGIN dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE', estimate_percent=>null, cascade=>true, method_opt=> 'FOR ALL COLUMNS SIZE 1'); END;

*  
ERROR at line 1:  
ORA-00904: : invalid identifier  
ORA-06512: at "SYS.DBMS_STATS", line 17806  
ORA-06512: at "SYS.DBMS_STATS", line 17827  
ORA-06512: at line 1
```

Fixed In Oracle11g Release 2
Creation On Demand Segments

- Prior to 11g Release 2, when a segment is created, it’s allocated at least 1 initial extent
- However, many large applications create many segments that are not actually used (e.g. SAP)
- Many tables can have many associated indexes which are likewise never used
- This can result in a large amount of essentially wasted storage overall
SQL> create table empty (a number, b number, c number, d number, e number);
Table created.

SQL> create index empty_a_i on empty(a);
Index created.

SQL> create index empty_b_i on empty(b);
Index created.

SQL> create index empty_c_i on empty(c);
Index created.

SQL> create index empty_d_i on empty(d);
Index created.

SQL> create index empty_e_i on empty(e);
Index created.

SQL> select segment_name, blocks, bytes, extents from dba_segments where segment_name like 'EMPTY%';

SEGMENT_NAME     BLOCKS      BYTES    EXTENTS
---------------  -----------  -------  ------
EMPTY               128    1048576          1
EMPTY_A_I           128    1048576          1
EMPTY_B_I           128    1048576          1
EMPTY_C_I           128    1048576          1
EMPTY_D_I           128    1048576          1
EMPTY_E_I           128    1048576          1

6 rows selected.
Creation On Demand Segments

However, create the same segments in Oracle 11g Release 2 and no storage is allocated at all ...

```sql
SQL> create table empty (a number, b number, c number, d number, e number);
Table created.

SQL> create index empty_a_i on empty(a);
Index created.

SQL> create index empty_b_i on empty(b);
Index created.

SQL> create index empty_c_i on empty(c);
Index created.

SQL> create index empty_d_i on empty(d);
Index created.

SQL> create index empty_e_i on empty(e);
Index created.

SQL> select segment_name, blocks, bytes, extents from dba_segments where segment_name like 'EMPTY%';
no rows selected
```
Creation On Demand Segments

If we now create another table with lots of rows ...

SQL> create table bowie as select rownum id, 'BOWIE' name from dual connect by level <= 1000000;
Table created.

SQL> create index bowie_id_i on bowie(id);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE', cascade=>true, estimate_percent=>null, method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'EMPTY', cascade=>true, estimate_percent=>null, method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
SQL> select * from bowie, empty where bowie.id=empty.a and bowie.id = 42;
no rows selected

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>76</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>MERGE JOIN CARTESIAN</td>
<td></td>
<td>1</td>
<td>76</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>EMPTY</td>
<td>1</td>
<td>65</td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*  3</td>
<td>INDEX RANGE SCAN</td>
<td>EMPTY_A_I</td>
<td>1</td>
<td></td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>4</td>
<td>BUFFER SORT</td>
<td></td>
<td>1</td>
<td>11</td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BOWIE</td>
<td>1</td>
<td>11</td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*  6</td>
<td>INDEX RANGE SCAN</td>
<td>BOWIE_ID_I</td>
<td>1</td>
<td></td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

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

September 2010
Creation On Demand Segments

If we insert the first row in a pre-Oracle11g Rel 2 table:

```
SQL> insert into empty2 (a, b) values (1,1);
1 row created.
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INSERT STATEMENT</td>
<td></td>
<td>1</td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>LOAD TABLE CONVENTIONAL</td>
<td>EMPTY2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics

- 3 recursive calls
- 10 db block gets
- 6 consistent gets
- 0 physical reads
- 1056 redo size
- 389 bytes sent via SQL*Net to client
- 322 bytes received via SQL*Net from client
- 3 SQL*Net roundtrips to/from client
- 2 sorts (memory)
- 0 sorts (disk)
- 1 rows processed

Overheads are minimal ...
However, when we insert the first row in an Oracle11g Rel 2 table

```sql
SQL> insert into empty (a, b) values (1,1);
1 row created.
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INSERT STATEMENT</td>
<td></td>
<td>1</td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>LOAD TABLE CONVENTIONAL</td>
<td>EMPTY</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics

```
759  recursive calls
129  db block gets
133  consistent gets
  3  physical reads
21232 redo size
389  bytes sent via SQL*Net to client
321  bytes received via SQL*Net from client
  3  SQL*Net roundtrips to/from client
  2  sorts (memory)
  0  sorts (disk)
  1  rows processed
```
The first row creates all dependent segments, even if indexes are not populated.
Quotas With Creation On Demand Segments

Prior to 11g R2, could not create a segment in a tablespace without sufficient quotas ...

```
SQL> create user muse identified by muse default tablespace user_data temporary tablespace temp;
User created.

SQL> grant create session, create table to muse;
Grant succeeded.

SQL> connect muse/muse;
Connected.

SQL> create table fred (id number primary key using index (create index fred_pk on fred(id) tablespace user_data), name varchar2(20));
create table fred (id number primary key using index (create index fred_pk on fred(id) tablespace user_data), name varchar2(20))
*  
ERROR at line 1:
ORA-01950: no privileges on tablespace 'USER_DATA'
```
Quotas With Creation On Demand Segments

However, in 11g R2, you can with quotas only enforced when data is actually inserted ...

SQL> create user muse identified by muse default tablespace user_data temporary tablespace temp;
User created.

SQL> grant create session, create table to muse;
Grant succeeded.

SQL> connect muse/muse
Connected.

SQL> create table fred (id number primary key using index (create index fred_pk on fred(id) tablespace user_data), name varchar2(20));
Table created.

SQL> insert into fred values (1, 'BOWIE');
insert into fred values (1, 'BOWIE')
* ERROR at line 1:
ORA-01950: no privileges on tablespace 'USER_DATA'

September 2010 ©Richard Foote - Indexing New Features: Oracle 11g Release 1 and Release 2
Zero Sized Unusable Indexes

Prior to Oracle11g Release 2

```sql
SQL> create table bowie as select rownum id, 'BOWIE' name from dual connect by level <= 1000000;
Table created.

SQL> create index bowie_id_i on bowie(id);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE', cascade=> true,
estimate_percent=> null, method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.

SQL> alter index bowie_id_i unusable;
Index altered.

SQL> select index_name, blevel, leaf_blocks, num_rows, status, dropped from dba_indexes where index_name = 'BOWIE_ID_I';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>BLEVEL</th>
<th>LEAF_BLOCKS</th>
<th>NUM_ROWS</th>
<th>STATUS</th>
<th>DRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ID_I</td>
<td>2</td>
<td>2226</td>
<td>1000000</td>
<td>UNUSABLE</td>
<td>NO</td>
</tr>
</tbody>
</table>

SQL> select segment_name, bytes, blocks, extents from dba_segments where segment_name = 'BOWIE_ID_I';

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>BYTES</th>
<th>BLOCKS</th>
<th>EXTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ID_I</td>
<td>18874368</td>
<td>2304</td>
<td>18</td>
</tr>
</tbody>
</table>
```
### Zero Sized Unusable Indexes

**Oracle11g Release 2**

SQL> create table bowie as select rownum id, 'BOWIE' name from dual connect by level <= 1000000;
Table created.

SQL> create index bowie_id_i on bowie(id);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE', cascade=> true, estimate_percent=> null, method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.

SQL> alter index bowie_id_i unusable;
Index altered.

SQL> select index_name, blevel, leaf_blocks, num_rows, status, dropped from dba_indexes where index_name = 'BOWIE_ID_I';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>BLEVEL</th>
<th>LEAF_BLOCKS</th>
<th>NUM_ROWS</th>
<th>STATUS</th>
<th>DRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ID_I</td>
<td>2</td>
<td>2226</td>
<td>1000000</td>
<td>UNUSABLE</td>
<td>NO</td>
</tr>
</tbody>
</table>

SQL> select segment_name, bytes, blocks, extents from dba_segments where segment_name = 'BOWIE_ID_I';

no rows selected

Oracle automatically drops the storage associated with such unusable indexes.
Zero Sized Unusable Indexes

Create and populate a simple partitioned table

SQL> CREATE TABLE big_album_sales(id number, album_id number, country_id number, release_date date, total_sales number) PARTITION BY RANGE (release_date)
(PARTITION ALBUMS_2006 VALUES LESS THAN (TO_DATE('01-JAN-2007', 'DD-MON-YYYY')),
PARTITION ALBUMS_2007 VALUES LESS THAN (TO_DATE('01-JAN-2008', 'DD-MON-YYYY')),
PARTITION ALBUMS_2008 VALUES LESS THAN (TO_DATE('01-JAN-2009', 'DD-MON-YYYY')),
PARTITION ALBUMS_2009 VALUES LESS THAN (MAXVALUE));

Table created.

SQL> INSERT INTO big_album_sales SELECT rownum, mod(rownum,5000)+1, mod(rownum,100)+1, sysdate-mod(rownum,2000), ceil(dbms_random.value(1,500000)) FROM dual CONNECT BY LEVEL <= 1000000;

1000000 rows created.

SQL> commit;

Commit complete.
Zero Sized Unusable Indexes

Create a non-partition index, a global index and a local index on the partitioned table

SQL> CREATE INDEX big_album_tot_sales_i ON big_album_sales(total_sales);
Index created.

SQL> CREATE INDEX big_album_country_id_i ON big_album_sales(country_id)
    2 GLOBAL PARTITION BY RANGE (country_id)
    3 (PARTITION TS1 VALUES LESS THAN (26),
    4 PARTITION TS2 VALUES LESS THAN (51),
    5 PARTITION TS3 VALUES LESS THAN (76),
    6 PARTITION TS4 VALUES LESS THAN (MAXVALUE));
Index created.

SQL> CREATE INDEX big_album_album_id_i ON big_album_sales(album_id) LOCAL;
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=> 'BOWIE', tabname=> 'BIG_ALBUM_SALES',
estimate_percent=> null, method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
Zero Sized Unusable Indexes

SQL> ALTER TABLE big_album_sales SPLIT PARTITION ALBUMS_2009
   2 AT (TO_DATE('01-JAN-2010', 'DD-MON-YYYY'))
   3 INTO (PARTITION ALBUMS_2009, PARTITION ALBUMS_2010);

Table altered.

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG_ALBUM_TOT_SALES_I</td>
<td>UNUSABLE</td>
</tr>
<tr>
<td>BIG_ALBUM_COUNTRY_ID_I</td>
<td>N/A</td>
</tr>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>N/A</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>PARTITION_NAME</th>
<th>STATUS</th>
<th>LEAF_BLOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>ALBUMS_2006</td>
<td>USABLE</td>
<td>807</td>
</tr>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>ALBUMS_2007</td>
<td>USABLE</td>
<td>381</td>
</tr>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>ALBUMS_2008</td>
<td>USABLE</td>
<td>383</td>
</tr>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>ALBUMS_2009</td>
<td>UNUSABLE</td>
<td>807</td>
</tr>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>ALBUMS_2010</td>
<td>UNUSABLE</td>
<td>807</td>
</tr>
<tr>
<td>BIG_ALBUM_COUNTRY_ID_I</td>
<td>TS1</td>
<td>UNUSABLE</td>
<td>629</td>
</tr>
<tr>
<td>BIG_ALBUM_COUNTRY_ID_I</td>
<td>TS2</td>
<td>UNUSABLE</td>
<td>629</td>
</tr>
<tr>
<td>BIG_ALBUM_COUNTRY_ID_I</td>
<td>TS3</td>
<td>UNUSABLE</td>
<td>629</td>
</tr>
<tr>
<td>BIG_ALBUM_COUNTRY_ID_I</td>
<td>TS4</td>
<td>UNUSABLE</td>
<td>629</td>
</tr>
</tbody>
</table>
Zero Sized Unusable Indexes

All unusable index partitions from both the global and local index no longer have allocated storage.

However, the unusable non-partitioned index segment has not been dropped ...

```
SQL> select segment_name, partition_name, bytes, blocks from dba_segments where segment_name like 'BIG_ALBUM_%' and segment_type like 'INDEX%';
```

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>PARTITION_NAME</th>
<th>BYTES</th>
<th>BLOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>ALBUMS_2006</td>
<td>7340032</td>
<td>896</td>
</tr>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>ALBUMS_2007</td>
<td>3145728</td>
<td>384</td>
</tr>
<tr>
<td>BIG_ALBUM_ALBUM_ID_I</td>
<td>ALBUMS_2008</td>
<td>4194304</td>
<td>512</td>
</tr>
<tr>
<td>BIG_ALBUM_TOT_SALES_I</td>
<td></td>
<td>23068672</td>
<td>2816</td>
</tr>
</tbody>
</table>

All unusable index partitions from both the global and local index no longer have allocated storage.
However, the unusable non-partitioned index segment has not been dropped ...

```
SQL> select segment_name, partition_name, bytes, blocks from dba_segments where segment_name like 'BIG_ALBUM_%' and segment_type like 'INDEX%';
```
Zero Sized Unusable Indexes

Can use zero sized unusable indexes to your advantage to index only useful portions of a table. Most data here is processed:

```sql
SQL> create table bowie_stuff (id number, processed varchar2(10));
Table created.

SQL> insert into bowie_stuff select rownum, 'YES' from dual connect by level <= 1000000;
1000000 rows created.

SQL> commit;
Commit complete.

SQL> update bowie_stuff set processed = 'NO' where id in (999990, 999992, 999994, 999996, 999998);
5 rows updated.

SQL> commit;
Commit complete.

SQL> create index bowie_stuff_i on bowie_stuff(processed) pctfree 0;
Index created.
```
Zero Sized Unusable Indexes

SQL> select index_name, leaf_blocks from dba_indexes where index_name = 'BOWIE_STUFF_I';

INDEX_NAME                     LEAF_BLOCKS
-------------------------------  ------------
BOWIE_STUFF_I                   1877

SQL> select segment_name, blocks from dba_segments where segment_name = 'BOWIE_STUFF_I';

SEGMENT_NAME             BLOCKS
-------------------------  --------
BOWIE_STUFF_I            1920

SQL> exec dbms_stats.gather_table_stats(ownname=>'BOWIE', tabname=>'BOWIE_STUFF', estimate_percent=>null, cascade=> true, method_opt=> 'FOR ALL COLUMNS SIZE 1');

PL/SQL procedure successfully completed.

SQL> exec dbms_stats.gather_table_stats(ownname=>'BOWIE', tabname=>'BOWIE_STUFF', estimate_percent=>null, method_opt=> 'FOR COLUMNS PROCESSED SIZE 5');

PL/SQL procedure successfully completed.
Zero Sized Unusable Indexes

SQL> select * from bowie_stuff where processed = 'NO';

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>5</td>
<td>40</td>
<td>4 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BOWIE_STUFF</td>
<td>5</td>
<td>40</td>
<td>4 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BOWIE_STUFF_I</td>
<td>5</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

- 0 recursive calls
- 0 db block gets
- 6 consistent gets
- 0 physical reads
- 0 redo size
- 540 bytes sent via SQL*Net to client
- 396 bytes received via SQL*Net from client
- 2 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 5 rows processed
However in 11g R2, if we now recreate the index as a partitioned index with only the “useful” portion of the index usable ...

SQL> drop index bowie_stuff_i;
Index dropped.

SQL> create index bowie_stuff_i on bowie_stuff(processed)
   2  global partition by range (processed)
   3  (partition not_processed_part values less than ('YES'),
   4  partition processed_part values less than (MAXVALUE))
   5  unusable;
Index created.

SQL> alter index bowie_stuff_i rebuild partition not_processed_part;
Index altered.
Zero Sized Unusable Indexes

We now only use a fraction of the storage for the index and the “useful” portion of the indexed data is just a single leaf block in size ...

```sql
SQL> select index_name, partition_name, leaf_blocks from dba_ind_partitions where index_name = 'BOWIE_STUFF_I';

INDEX_NAME           PARTITION_NAME       LEAF_BLOCKS
--------------------  -------------------  ----------
BOWIE_STUFF_I        PROCESSED_PART       0
BOWIE_STUFF_I        NOT_PROCESSED_PART   1

SQL> select segment_name, partition_name, blocks from dba_segments where segment_name = 'BOWIE_STUFF_I';

SEGMENT_NAME         PARTITION_NAME       BLOCKS
-------------------  -------------------  ----
BOWIE_STUFF_I        NOT_PROCESSED_PART   8
```
Zero Sized Unusable Indexes

SQL> select * from bowie_stuff where processed = 'NO';

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>5</td>
<td>45</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td>5</td>
<td>45</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BOWIE_STUFF</td>
<td>5</td>
<td>45</td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 3</td>
<td>INDEX RANGE SCAN</td>
<td>BOWIE_STUFF_I</td>
<td>5</td>
<td></td>
<td>1 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics

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

Note: The query itself is also more efficient with consistent gets reduced from 6 down to 4 ...
SQL> create table radiohead (id number constraint radiohead_pk_i primary key using index (create unique index radiohead_pk_i on radiohead(id)), name varchar2(20));
Table created.

SQL> select index_name, uniqueness, table_name from dba_indexes where index_name = 'RADIOHEAD_PK_I';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>UNIQUENESS</th>
<th>TABLE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>RADIOHEAD_PK_I</td>
<td>UNIQUE</td>
<td>RADIOHEAD</td>
</tr>
</tbody>
</table>

SQL> insert into radiohead select rownum, 'OK COMPUTER' from dual connect by level <= 10;
10 rows created.

SQL> commit;
Commit complete.

SQL> insert into radiohead select rownum, 'OK COMPUTER' from dual connect by level <= 12;
insert into radiohead select rownum, 'OK COMPUTER' from dual connect by level <= 12
*  
ERROR at line 1:
ORA-00001: unique constraint (BOWIE.RADIOHEAD_PK_I) violated
With the new Oracle11g Rel. 1 hint, duplicate violation rows are automatically ignored.

```sql
SQL> insert /*+ ignore_row_on_dupkey_index(radiohead,radiohead_pk_i) */ into radiohead select rownum, 'OK COMPUTER' from dual connect by level <= 12;
2 rows created.

SQL> insert /*+ ignore_row_on_dupkey_index(radiohead(id)) */ into radiohead select rownum, 'OK COMPUTER' from dual connect by level <= 13;
1 row created.

SQL> commit;
Commit complete.

SQL> select * from radiohead;

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>2</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>3</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>4</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>5</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>6</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>7</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>8</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>9</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>10</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>11</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>12</td>
<td>OK COMPUTER</td>
</tr>
<tr>
<td>13</td>
<td>OK COMPUTER</td>
</tr>
</tbody>
</table>

13 rows selected.
```
The index must be Unique for the hint to be valid ...

```
SQL> create table radiohead (id number constraint radiohead_pk_i primary key using index (create index radiohead_pk_i on radiohead(id)), name varchar2(20));
Table created.
SQL> insert into radiohead select rownum, 'OK COMPUTER' from dual connect by level <= 10;
10 rows created.
SQL> commit;
Commit complete.
SQL> select index_name, uniqueness, table_name from dba_indexes where index_name='RADIOHEAD_PK_I';
<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>UNIQUENESS</th>
<th>TABLE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>RADIOHEAD_PK_I</td>
<td>NONUNIQUE</td>
<td>RADIOHEAD</td>
</tr>
</tbody>
</table>
SQL> insert /*+ ignore_row_on_dupkey_index(radiohead,radiohead_pk_i) */ into radiohead select rownum, 'OK COMPUTER' from dual connect by level <= 12;
insert /*+ ignore_row_on_dupkey_index(radiohead,radiohead_pk_i) */ into radiohead select rownum, 'OK COMPUTER' from dual connect by level <= 12 *
ERROR at line 1:
ORA-38913: Index specified in the index hint is invalid
```
The UPDATE statement is not allowed with this hint ...

```
SQL> update /*+ ignore_row_on_dupkey_index(radiohead,radiohead_pk_i) */ radiohead set id = 13 where id = 3;
update /*+ ignore_row_on_dupkey_index(radiohead,radiohead_pk_i) */ radiohead set id = 13 where id = 3
*  
ERROR at line 1:
ORA-38917: IGNORE_ROW_ON_DUPKEY_INDEX hint disallowed for this operation
```
Oracle11g Release 1 introduces a new “FAST”, more efficient VALIDATE STRUCTURE command option

Will identified a corruption exists but no specific details about the corruption

However, initial investigation with FAST option has sometimes yielded “disappointing” results ...
ANALYZE VALIDATE STRUCTURE FAST

If we trace a session using just the VALIDATE STRUCTURE CASCADE option:

```
analyze table log_entries validate structure cascade
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>37.56</td>
<td>127.03</td>
<td>44679</td>
<td>14300957</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**total** | **2** | **37.57** | **127.10** | **44679** | **14300957** | **2** | **0** |

We notice that it causes a massive number of query based I/Os
ANALYZE VALIDATE STRUCTURE FAST

If we trace a session using the new VALIDATE STRUCTURE CASCADE FAST option:

```sql
analyze table log_entries validate structure cascade fast

select /*+ full(LOG_ENTRIES) */ ORA_HASH(DATE_TIME_ACTIONED || rowid)
from LOG_ENTRIES MINUS select /*+ index_ffs(LOG_ENT_DATE_ACT_I) */
    ORA_HASH(DATE_TIME_ACTIONED || rowid) from LOG_ENTRIES
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>5.93</td>
<td>15.49</td>
<td>13022</td>
<td>13190</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| total  | 3     | 5.94 | 15.50   | 13022| 13190 | 0       | 0    |

```sql
select /*+ full(LOG_ENTRIES) */ ORA_HASH(CASE_ID || DATE_TIME_ADDED || rowid)
from LOG_ENTRIES MINUS select /*+ index_ffs(LOG_ENT_DATE_ADDED_I) */
    ORA_HASH(CASE_ID || DATE_TIME_ADDED || rowid) from LOG_ENTRIES
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>1</td>
<td>6.22</td>
<td>7.64</td>
<td>1345</td>
<td>3148</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| total  | 3     | 6.23 | 7.65    | 1345 | 3148  | 0       | 0    |

We notice it utilises the ORA_HASH function to compare differences between a full table scan and a fast full index scan.
ANALYZE VALIDATE STRUCTURE FAST

If we look at the overall total resources:

<table>
<thead>
<tr>
<th>OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Parse</td>
</tr>
<tr>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
</tr>
<tr>
<td>total</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Parse</td>
</tr>
<tr>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
</tr>
<tr>
<td>total</td>
</tr>
</tbody>
</table>

We notice it’s using significantly less I/Os but much more CPU

Overall, this can sometimes result in the overall elapsed times actually being greater with FAST than without ...
Function-Based Indexes

```sql
SQL> CREATE TABLE func_tab AS SELECT ROWNUM id, 'DAVID BOWIE '||
ceil(dbms_random.value(0,10000000)) name FROM DUAL CONNECT BY LEVEL <= 100000;
Table created.

SQL> INSERT INTO func_tab VALUES (100001, 'Ziggy');
1 row created.

SQL> INSERT INTO func_tab VALUES (100002, 'ZIGGY');
1 row created.

SQL> INSERT INTO func_tab VALUES (100003, 'ZiGgY');
1 row created.

SQL> commit;
Commit complete.

SQL> CREATE INDEX func_tab_name_i ON func_tab(name);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>'BOWIE', tabname=>
'FUNC_TAB',cascade=> true, estimate_percent=>null, method_opt=> 'FOR ALL COLUMNS
SIZE 1');
PL/SQL procedure successfully completed.
```
Function-Based Indexes

The use of the UPPER function negates the use of the index.

SQL> SELECT * FROM func_tab WHERE UPPER(name) = 'ZIGGY';

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>Ziggy</td>
</tr>
<tr>
<td>100002</td>
<td>ZIGGY</td>
</tr>
<tr>
<td>100003</td>
<td>ZiGgY</td>
</tr>
</tbody>
</table>

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL</td>
<td>FUNC_TAB</td>
<td>1000</td>
<td>24000</td>
<td>89 (5)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

1 recursive calls
0 db block gets
421 consistent gets
0 physical reads
0 redo size
530 bytes sent via SQL*Net to client
396 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
3 rows processed

Note also that the cardinality estimate of 1000 rows is way off ...
Function-Based Indexes

SQL> CREATE INDEX func_tab_upper_name_i ON func_tab(UPPER(name))
   COMPUTE STATISTICS;
Index created.

SQL> SELECT * FROM func_tab WHERE UPPER(name) = 'ZIGGY';

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>Ziggy</td>
</tr>
<tr>
<td>100002</td>
<td>ZIGGY</td>
</tr>
<tr>
<td>100003</td>
<td>ZiGgY</td>
</tr>
</tbody>
</table>

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>84      (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>FUNC_TAB</td>
<td>1000</td>
<td>24000</td>
<td>84      (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>FUNC_TAB_UPPER_NAME_I</td>
<td>400</td>
<td></td>
<td>3        (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

The cardinality estimate is still way off at 1000 rows when only 3 rows are actually returned ...

Function-based index has fortunately been used but such wrong cardinality estimates can potentially result in disastrous execution plans.
Virtual Columns

• When a Function-Based Index is created, Oracle creates a “Virtual” column for the table.
• Prior to 11g, these Virtual columns were “Hidden”.
• These columns are used to store column statistics to be subsequently used by the CBO.
• The table, not the index, needs to have statistics collected to populate Virtual columns.
### Virtual Columns

**SQL Query Examples:**

```sql
SQL> SELECT table_name, column_name, num_distinct, density
FROM dba_tab_columns WHERE table_name = 'FUNC_TAB';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>DENSITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_TAB</td>
<td>ID</td>
<td>100003</td>
<td>9.9997E-06</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>NAME</td>
<td>99495</td>
<td>.000010051</td>
</tr>
</tbody>
</table>

SQL> SELECT table_name, column_name, num_distinct, density, virtual_column, hidden_column
FROM dba_tab_cols WHERE table_name = 'FUNC_TAB';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>DENSITY</th>
<th>VIR</th>
<th>HID</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_TAB</td>
<td>SYS_NC00003$</td>
<td>99495</td>
<td>.000010051</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>NAME</td>
<td>99495</td>
<td>.000010051</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>ID</td>
<td>100003</td>
<td>9.9997E-06</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

SQL> SELECT column_name, num_distinct, density, avg_col_len
FROM dba_tab_col_statistics WHERE table_name = 'FUNC_TAB';

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>DENSITY</th>
<th>AVG_COL_LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>100003</td>
<td>9.9997E-06</td>
<td>5</td>
</tr>
<tr>
<td>NAME</td>
<td>99495</td>
<td>.000010051</td>
<td>18</td>
</tr>
</tbody>
</table>
```

*September 2010*
Virtual Columns

SQL> exec dbms_stats.gather_table_stats(ownname=>'BOWIE', tabname=>'FUNC_TAB', cascade=> true, estimate_percent=>null, method_opt=>'FOR ALL HIDDEN COLUMNS SIZE 1');

PL/SQL procedure successfully completed.

SQL> SELECT table_name, column_name, num_distinct, density, virtual_column, hidden_column FROM dba_tab_cols WHERE table_name = 'FUNC_TAB';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>DENSITY</th>
<th>VIR</th>
<th>HID</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_TAB</td>
<td>SYS_NC00003$</td>
<td>99493</td>
<td>.000010051</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>NAME</td>
<td>99495</td>
<td>.000010051</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>ID</td>
<td>100003</td>
<td>9.9997E-06</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

Note: the virtual statistics are now populated ...

©Richard Foote - Indexing New Features: Oracle 11g Release 1 and Release 2
SQL> SELECT * FROM func_tab WHERE UPPER(name) = 'ZIGGY';

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>Ziggy</td>
</tr>
<tr>
<td>100002</td>
<td>ZIGGY</td>
</tr>
<tr>
<td>100003</td>
<td>ZiGgY</td>
</tr>
</tbody>
</table>

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>44</td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>FUNC_TAB</td>
<td>1</td>
<td>44</td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>FUNC_TAB_UPPER_NAME_I</td>
<td>1</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Note: statistics are now much more accurate, remembering that all rows are unique, except the “ZIGGY” rows ...
11g Virtual Columns

• Prior to 11g, virtual columns were created through Function-Based Indexes
• Virtual columns useful due to statistics but had to create index even if it wasn’t required
• Virtual columns were “hidden” and had to be manually derived in queries
• 11g now allows Virtual Columns to be defined and visible in a table definition
Can define a visible, virtual column to a table ...
11g Virtual Columns

SQL> SELECT table_name, column_name, num_distinct, density 
FROM dba_tab_columns WHERE table_name = 'FUNC_TAB';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>DENSITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_TAB</td>
<td>ID</td>
<td>100003</td>
<td>9.9997E-06</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>NAME</td>
<td>99495</td>
<td>.000010051</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>UPPER_NAME</td>
<td>99493</td>
<td>.000010051</td>
</tr>
</tbody>
</table>

SQL> SELECT table_name, column_name, num_distinct, density, virtual_column, 
    hidden_column FROM dba_tab_cols WHERE table_name = 'FUNC_TAB';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>DENSITY</th>
<th>VIR</th>
<th>HID</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_TAB</td>
<td>UPPER_NAME</td>
<td>99493</td>
<td>.000010051</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>NAME</td>
<td>99495</td>
<td>.000010051</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>FUNC_TAB</td>
<td>ID</td>
<td>100003</td>
<td>9.9997E-06</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

SQL> SELECT column_name, num_distinct, density, avg_col_len FROM dba_tab_col_statistics 
WHERE table_name = 'FUNC_TAB';

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>DENSITY</th>
<th>AVG_COL_LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPER_NAME</td>
<td>99493</td>
<td>.000010051</td>
<td>18</td>
</tr>
<tr>
<td>NAME</td>
<td>99495</td>
<td>.000010051</td>
<td>18</td>
</tr>
<tr>
<td>ID</td>
<td>100003</td>
<td>9.9997E-06</td>
<td>5</td>
</tr>
</tbody>
</table>
Partial block dump of table

```
block_row_dump:
tab 0, row 0, @0x1f69
tl: 23 fb: --H-FL-- lb: 0x0  cc: 2
col 0: [ 3] c2 06 58
col 1: [15] 44 41 56 49 44 20 42 4f 57 49 45 20 35 38 37
tab 0, row 1, @0x1f52
tl: 23 fb: --H-FL-- lb: 0x0  cc: 2
col 0: [ 3] c2 06 59
col 1: [15] 44 41 56 49 44 20 42 4f 57 49 45 20 35 38 38
tab 0, row 2, @0x1f3b
tl: 23 fb: --H-FL-- lb: 0x0  cc: 2
col 0: [ 3] c2 06 5a
col 1: [15] 44 41 56 49 44 20 42 4f 57 49 45 20 35 38 39
```

Note each row only has the 2 column values physically stored in the table
SQL> SELECT * FROM func_tab WHERE UPPER(name) = 'ZIGGY';

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>UPPER_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>Ziggy</td>
<td>ZIGGY</td>
</tr>
<tr>
<td>100002</td>
<td>ZIGGY</td>
<td>ZIGGY</td>
</tr>
<tr>
<td>100003</td>
<td>ZiGgY</td>
<td>ZIGGY</td>
</tr>
</tbody>
</table>

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>44</td>
<td>89 (5)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>FUNC_TAB</td>
<td>1</td>
<td>44</td>
<td>89 (5)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

- 0 recursive calls
- 0 db block gets
- 421 consistent gets
- 0 physical reads
- 0 redo size
- 596 bytes sent via SQL*Net to client
- 396 bytes received via SQL*Net from client
- 2 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 3 rows processed
11g Virtual Columns

SQL> create index upper_name_idx on func_tab(upper(name));
create index upper_name_idx on func_tab(upper(name))
  *

ERROR at line 1:
ORA-54018: A virtual column exists for this expression

SQL> create index upper_name_idx on func_tab(upper_name);

Index created.

SQL> SELECT index_name, index_type FROM user_indexes WHERE index_name = 'UPPER_NAME_IDX';

INDEX_NAME              INDEX_TYPE
------------------------ ------------------------------
UPPER_NAME_IDX           FUNCTION-BASED NORMAL

SQL> SELECT index_name, column_expression FROM user_ind_expressions WHERE index_name = 'UPPER_NAME_IDX';

INDEX_NAME              COLUMN_EXPRESSION
------------------------ --------------------
UPPER_NAME_IDX           UPPER("NAME")
11g Virtual Columns

SQL> select * from func_tab where upper_name = 'ZIGGY';

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>UPPER_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>Ziggy</td>
<td>ZIGGY</td>
</tr>
<tr>
<td>100002</td>
<td>ZIGGY</td>
<td>ZIGGY</td>
</tr>
<tr>
<td>100003</td>
<td>ZiGgY</td>
<td>ZIGGY</td>
</tr>
</tbody>
</table>

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>45</td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>FUNC_TAB</td>
<td>1</td>
<td>45</td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>UPPER_NAME_IDX</td>
<td>1</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
6 consistent gets
0 physical reads
0 redo size
336 bytes sent via SQL*Net to client
247 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
3 rows processed
Virtual columns can be used where non-virtual columns are used, including as the partitioning key in a partitioned table:

```
SQL> CREATE TABLE album_sales(id number, album_id number, country_id number,
2  release_date date, total_sales number, total_value as (total_sales*10.95)) PARTITION BY RANGE (total_value)
3  (PARTITION ALBUMS_POOR VALUES LESS THAN (100000),
4  PARTITION ALBUMS_500000 VALUES LESS THAN (500001),
5  PARTITION ALBUMS_1000000 VALUES LESS THAN (1000001),
6  PARTITION ALBUMS_2000000 VALUES LESS THAN (2000001),
7  PARTITION ALBUMS_3000000 VALUES LESS THAN (3000001),
8  PARTITION ALBUMS_4000000 VALUES LESS THAN (4000001),
9  PARTITION ALBUMS_5000000 VALUES LESS THAN (5000001),
10 PARTITION ALBUMS_6000000 VALUES LESS THAN (6000001),
11 PARTITION ALBUMS_7000000 VALUES LESS THAN (7000001),
12 PARTITION ALBUMS_8000000 VALUES LESS THAN (8000001),
13 PARTITION ALBUMS_9000000 VALUES LESS THAN (9000001),
14 PARTITION ALBUMS_BESTSELLERS VALUES LESS THAN (MAXVALUE));
```

Table created.
Introduction To Reverse Key Indexes: Part III (A Space Oddity) January 18, 2008

Posted by Richard Foote in Oracle Indexes
9 comments

A possibly significant difference between a Reverse and a Non-Reverse index is the manner in which space is used in each index and the type of block splitting that takes place.

Most Reverse Key Indexes are created to resolve contention issues as a result of monotonically increasing values. As monotonically increasing values get inserted, each value is greater than all previous values (providing there are no outer values present) and so fill the “right-most” leaf block. If the “right-most” block is filled by the maximum current value in the index, Oracle performs 90-10 block splits meaning that full index blocks are left behind in the index structure. Assuming no deletes or updates, the index should have virtually 100% used space.

However, it’s equivalent Reverse Key index will have the values reversed and dispersed evenly throughout the index structure. As index blocks fill, there will be a very remote chance of it being due to the maximum indexed value and 50-50 block splits will result. The PCT_USED is likely therefore to be significantly less, averaging approximately 70-75% over time.

Therefore, for indexes with no deletes, a Reverse Key index is likely to be less efficient from a space usage point of view.

However, if there are deletes, the story may differ.

Deleted space can be reused if an insert is subsequently made into an index block with deleted entries or if a leaf block is totally emptied. However, if a leaf block contains any non-deleted entries and if subsequent inserts don’t hit the leaf block, then the deleted space can not be reused. As monotonically increasing values in a non-reverse index only ever insert into the “right-most” leaf block, it won’t be able to reuse deleted space if leaf blocks are not totally emptied. Overtime, the number of such “almost but not quite empty” index leaf blocks may in some scenarios increase to significant levels and the index may continue to grow at a greater proportional rate than the table (where the space of space is set and controlled by the PCT_USED initialization parameter).
Unconference Oracle Indexes Q@A With Richard Foote
Mason Room Hotel Parc 55
10 am Tuesday 21 September 2010

Thank you 😊