Indexing and Clustering Data: Key To Developing High Performance & Scalable Apps

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 Consulting: https://richardfooteconsulting.com
- Spend as much free time as possible listening to the music of David Bowie!!
Richard Foote’s Oracle Blog

Introduction To Reverse Key Indexes: Part III (A Space Oddity)

January 18, 2008

Recent Posts
- Introduction To Reverse Key Indexes: Part III (A Space Oddity)
- Symposium Series #9 and #10
- Public Appearances
- Introduction To Reverse Key Indexes: Part II (Structured Indexes)
- Introduction To Reverse Key Indexes: Part I (Plain Old Indexes)
- Things You May Not Know About Indexes
- TOAST KISS Already!
- Introduction To Unique Indexes - Part II
- Differences Between Unique and Non-Unique Indexes (Part I)
- Local Index Issue With Partitioned Schema and Unique Key Constraints
- Basing Index Row getters of RMAN BACKUP TABLESPACE...
Oracle Indexing Internals and Best Practices 5 Day Webinars

8-12 October, 6-10 November 2018 (4 Hours Daily)

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

Examines most available index structures/options & discusses in considerable detail how indexes function, how/when they should be used & how they should be maintained. Also how indexes are costed & evaluated by the Cost Based Optimizer (CBO) & how appropriate data management practices are vital for an effective indexing strategy. Covers many useful tips and strategies to maximise the benefits of indexes on application/database performance & scalability.

Running between 10am-2pm Zurich Time 8\textsuperscript{th} – 12\textsuperscript{th} October 2018

richardfooteconsulting.com/indexing-webinar/
Relational Database: Tables/Indexes...

- Data store made of tables
- For faster performance, create indexes
- Sometimes, just creating an index doesn’t do the trick in relation to improving performance ...

© RICHARD FOOTE CONSULTING
Importance of Data Clustering

The importance of Data Clustering is relevant:

- On Generic Hardware vs. Oracle Engineered Systems
- On Premises vs. In The Cloud
- Self Managed vs. Autonomous Database
- Oracle Databases vs. Non Oracle DBs
Lots Of Users Complaining: Which Query Would You Focus On?

Query 1
- Query returns 10,000 rows
- Performs approx. 10,000 logical I/Os
- No physical I/Os reported
- Data returned in 0.5 second

vs.

Query 2
- Query returns 10,000 rows
- Performs approx. 10M logical I/Os
- Millions of physical I/Os reported
- Data returned in 100 seconds
Demo Setup

SQL> create table bowie (id number, code number, grade number, sales number, status varchar2(42), date_entered date, name1 varchar2(42), name2 varchar2(42), name3 varchar2(42), name4 varchar2(42), name5 varchar2(42));
Table created.

SQL> insert into bowie select rownum, mod(rownum,100), ceil(dbms_random.value(0,100)),
ceil(dbms_random.value(0,10000)), 'PROCESSED', sysdate-mod(rownum, 1825), 'DAVID BOWIE', 'ZIGGY STARDUST', 'THE THIN WHITE DUKE', 'ALADDIN SANE', 'MAJOR TOM' from dual connect by level <=1000000;
4000000 rows created.

SQL> insert into bowie select rownum+4000000, mod(rownum,100), ceil(dbms_random.value(0,100)),
ceil(dbms_random.value(0,10000)), 'PROCESSED', sysdate-mod(rownum, 1825), 'DAVID BOWIE', 'ZIGGY STARDUST', 'THE THIN WHITE DUKE', 'ALADDIN SANE', 'MAJOR TOM' from dual connect by level <=4000000;
4000000 rows created.

SQL> commit;
Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BOWIE');
PL/SQL procedure successfully completed.

SQL> select table_name, num_rows, blocks, round(blocks*8/1024) SIZE_IN_MB from user_tables
where table_name = 'BOWIE';

 TABLE_NAME  NUM_ROWS  BLOCKS  SIZE_IN_MB
----------  --------  ------  ----------
 BOWIE     8000000   121019    945
Query Based On Unique ID Column

```
SQL> select * from bowie where id=42;
Elapsed: 00:00:09.46

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>104</td>
<td>32897   (1)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>BOWIE</td>
<td>1</td>
<td>104</td>
<td>32897   (1)</td>
<td>00:00:02</td>
</tr>
</tbody>
</table>

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

  1 - filter("ID"=42)

Statistics
```
0  recursive calls
0  db block gets
121042  consistent gets
121020  physical reads
0  redo size
1311  bytes sent via SQL*Net to client
607  bytes received via SQL*Net from client
2  SQL*Net roundtrips to/from client
0  sorts (memory)
0  sorts (disk)
1  rows processed
Query Based On CODE Column (1%)

```sql
SQL> select * from bowie where code=42;
Elapsed: 00:00:10.17
```

**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>80000</td>
<td>8125K</td>
<td>32905   (1)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>*  1</td>
<td>TABLE ACCESS FULL</td>
<td>BOWIE</td>
<td>80000</td>
<td>8125K</td>
<td>32905   (1)</td>
<td>00:00:02</td>
</tr>
</tbody>
</table>
```

Predicate Information (identified by operation id):

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

**Statistics**

- 0 recursive calls
- 0 db block gets
- 121041 consistent gets
- 121020 physical reads
- 0 redo size
- 2149515 bytes sent via SQL*Net to client
- 772 bytes received via SQL*Net from client
- 17 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 80000 rows processed
Let’s Create Indexes To Improve Performance

```sql
SQL> alter table bowie add constraint bowie_id_pk primary key (id);
Table altered.

SQL> create index bowie_code_i on bowie(code);
Index created.

SQL> create index bowie_grade_i on bowie(grade);
Index created.
```
Query With Index On Unique ID Column

SQL> select * from bowie where id=42;
Elapsed: 00:00:00.00

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>104</td>
<td>3 (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>104</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX UNIQUE SCAN</td>
<td>BOWIE_ID_PK</td>
<td>1</td>
<td></td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("ID"=42)

Statistics

0 recursive calls
0 db block gets
4 consistent gets
0 physical reads
0 redo size
1171 bytes sent via SQL*Net to client
596 bytes received via SQL*Net from client
1 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed
Query With Index On CODE Column (1%): FTS !!

SQL> select * from bowie where code=42;
Elapsed: 00:00:09.92

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>80000</td>
<td>8125K</td>
<td>32905</td>
<td>00:00:02</td>
</tr>
<tr>
<td>*  1</td>
<td>TABLE ACCESS FULL</td>
<td>BOWIE</td>
<td>80000</td>
<td>8125K</td>
<td>32905</td>
<td>00:00:02</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

  1 - filter("CODE"=42)

Statistics

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursive calls</td>
<td>1</td>
</tr>
<tr>
<td>db block gets</td>
<td>0</td>
</tr>
<tr>
<td>consistent gets</td>
<td>121041</td>
</tr>
<tr>
<td>physical reads</td>
<td>121020</td>
</tr>
<tr>
<td>redo size</td>
<td>0</td>
</tr>
<tr>
<td>bytes sent via SQL*Net to client</td>
<td>2149515</td>
</tr>
<tr>
<td>bytes received via SQL*Net from client</td>
<td>772</td>
</tr>
<tr>
<td>SQL*Net roundtrips to/from client</td>
<td>17</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>80000</td>
</tr>
</tbody>
</table>
Most Common Solution: Index Hint

SQL> select /*+ index (bowie, bowie_code_i) */ * from bowie where code=42;
Elapsed: 00:00:11.73

Execution Plan
--------------------------------------------------------------------------------------------
| Id  | Operation                  | Name         | Rows  | Bytes | Cost (%CPU) | Time     |
--------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |              | 80000 | 8125K| 80188   (1) | 00:00:04 |
| 00:00:04 |
|   1 | TABLE ACCESS BY INDEX ROWID| BOWIE        | 80000 | 8125K| 80188   (1) | 00:00:04 |
|*  2 | INDEX RANGE SCAN           | BOWIE_CODE_I | 80000 |       |   159   (1) | 00:00:01 |
--------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
  2 - access("CODE"=42)

Statistics
-----------------------------------------------
 1 recursive calls
 0 db block gets
 80175 consistent gets
 81945 physical reads
 0 redo size
 8712544 bytes sent via SQL*Net to client
 772 bytes received via SQL*Net from client
 17 SQL*Net roundtrips to/from client
 0 sorts (memory)
 0 sorts (disk)
 80000 rows processed
Importance of Data Clustering
Why Is Index Ignored?

As Clustering Factor matches number of rows in table, need to effectively access a different block for each and every row accessed via the BOWIE_CODE_I index.

For a given CODE value, need to access $0.01 \times 8000000 = 80,000$ table blocks via single block reads.

The table has 121,019 blocks which can be accessed more efficiently via FTS multi-block reads.
Clustering Factor (CF)

- Vital statistic used by CBO to determine cost of index access / index efficiency
- Determines the relative order of the table in relation to the index
- CF value corresponds to likely physical I/Os or blocks visited during a full index scan (note same block could be visited many times)
- If the same block is read consecutively via index, Oracle assumes only the 1 physical I/O is necessary
- The better the CF, the more efficient the access via the corresponding index as less physical I/Os are likely
- “Good” CF generally has value closer to blocks in table
- “Bad” CF generally has a value closer to rows in table
How Does Oracle Calculate CF?

- Performs a full index scan (or estimate thereof)
- Examines each rowid value to determine if specific block referenced is the same block as the previous rowid (by default)
- If it differs, the CF is incremented by 1
- At the end of the full index scan, the final tally becomes the CF of the index
Indexes vs. FTS

Cost vs. Rows Returned

- Full Table Scan
- Index Scan

Note: Clustering Factor determines line gradient
Index With Poor CF
New Table Ordered/Clustered on CODE Column

SQL> create table ziggy as select * from bowie order by code;
Table created.

SQL> alter table ziggy add constraint ziggy_id_pk primary key (id);
Table altered.

SQL> create index ziggy_code_i on ziggy(code);
Index created.

SQL> create index ziggy_grade_i on ziggy(grade);
Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'ZIGGY', estimate_percent => 100, method_opt=>'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
As CODE Clustering Factor (CF) matches closely to number of blocks in table, each block only need be accessed once during an index scan. The CF on CODE is now “perfect”.

For a given CODE value, need to only access 0.01x121032 = 1,210 table blocks. So index is now some just 1.5% of previous cost.

Note: CF on ID column is now much worse than previously.
Index With Good CF

Index

Table Blocks
### Query on ID with Table Clustered on CODE Column

```sql
SQL> select * from ziggy where id=42;
```

**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>104</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ZIGGY</td>
<td>1</td>
<td>104</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX UNIQUE SCAN</td>
<td>ZIGGY_ID_PK</td>
<td>1</td>
<td></td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
```

Predicate Information (identified by operation id):

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

**Statistics**

- 0 recursive calls
- 0 db block gets
- 4 consistent gets  => No difference as clustering factor has no impact when selecting one row
- 0 physical reads
- 0 redo size
- 1171 bytes sent via SQL*Net to client
- 596 bytes received via SQL*Net from client
- 1 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 1 rows processed
Query on CODE on Table Clustered on CODE Column

SQL> select * from ziggy where code=42;

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>80000</td>
<td>8125K</td>
<td>1327   (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ZIGGY</td>
<td>80000</td>
<td>8125K</td>
<td>1327   (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>INDEX RANGE SCAN</td>
<td>ZIGGY_CODE_I</td>
<td>80000</td>
<td></td>
<td>159   (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("CODE"=42)

Statistics

0 recursive calls
0 db block gets
1402 consistent gets (previously 121041 with FTS, 80175 with index)
0 physical reads
0 redo size
8712544 bytes sent via SQL*Net to client
773 bytes received via SQL*Net from client
17 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
80000 rows processed
Clustering Factor: How It Can Be Impacted

• Index clustering is improved when data is inserted in the same order as index
• Therefore anything that impacts this ordering can impact the clustering factor of an index
  – Column order in index
  – Reverse Indexes
  – Freelists / Freelist Groups
  – Automatic Segment Space Management (ASSM)
Index With Good Clustering But Poor CF

Index

Table Blocks
Most Common Performance Issue I
Encounter!!

SQL> create table bowie_assm (id number, name varchar2(42));
Table created.

SQL> create sequence bowie_assm_seq order;
Sequence created.

SQL> create or replace procedure pop_bowie_assm as
  2  begin
  3  for i in 1..100000 loop
  4  insert into bowie_assm values (bowie_assm_seq.nextval, 'DAVID BOWIE');
  5  commit;
  6  end loop;
  7  end;
  8  /
Procedure created.

In (say) 3 separate concurrent sessions, exec pop_bowie_assm
Default Clustering Factor Terrible

SQL> create index bowie_assm_id_i on bowie_assm(id);

Index created.

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

PL/SQL procedure successfully completed.

SQL> SELECT t.table_name, i.index_name, t.blocks, t.num_rows, i.clustering_factor
    FROM user_tables t, user_indexes i
    WHERE t.table_name = i.table_name AND i.index_name='BOWIE_ASSM_ID_I';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ASSM</td>
<td>BOWIE_ASSM_ID_I</td>
<td>1000</td>
<td>300000</td>
<td>217236</td>
</tr>
</tbody>
</table>

Data “inserted” in ID order. Index has extremely poor CF despite well clustered data.
Selecting Less 0.13% of Data

SQL> select * from bowie_assm where id between 42 and 429;

388 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>389</td>
<td>6613</td>
<td>275 (2)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>BOWIE_ASSM</td>
<td>389</td>
<td>6613</td>
<td>275 (2)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - filter("ID"<=429 AND "ID">=42)

Statistics

0 recursive calls
0 db block gets
949 consistent gets
0 physical reads
0 redo size
4094 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)
388 rows processed
Myth: Rebuild Index With High CF

- Rebuilding index if CF is poor is common advice
- Unfortunately, as neither table nor index order changes, the net effect is “disappointing” (actually no effect whatsoever).
- To improve the CF, it's the **table** that must be rebuilt (and reordered)
- If table has multiple indexes, careful consideration needs to be given by which index to order table
- Pre-fetch index reads improves poor CF performance
- Rebuilding an index simply because it has a CF over a certain threshold is futile and a silly myth
Myth: Rebuild Index With High CF

- Simple experiment ...
- Pick any index with as “bad” a Clustering Factor as can be found
- Collect fresh statistics with 100% estimate
- Rebuild the index (without changing definition, such as Reverse)
- Check out the “new” Clustering Factor
- The Clustering Factor will remain unchanged
Let's Try To Rebuild Index To Improve CF

```
SQL> alter index bowie_assm_id_i rebuild;
Index altered.

SQL> SELECT t.table_name, i.index_name, t.blocks, t.num_rows, i.clustering_factor FROM user_tables t, user_indexes i WHERE t.table_name = i.table_name AND i.index_name='BOWIE_ASSM_ID_I';
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ASSM</td>
<td>BOWIE_ASSM_ID_I</td>
<td>1000</td>
<td>300000</td>
<td>217236</td>
</tr>
</tbody>
</table>

The Clustering Factor remains identical...
CF Improvement (TABLE_CACHED_BLOCKS)

- Bug 13262857 Enh: provide some control over DBMS_STATS index clustering factor computation INDEX
- Patch available for 11.1.0.7, 11.2.0.2, 11.2.0.3
- TABLE_CACHED_BLOCKS stats collection preference
- Can be set between 1 (default) and 255 to determine how many previous accessed blocks to ignore when incrementing CF
- Set preference via for example:
  - DBMS_STATS.SET_TABLE_PREFS,
  - DBMS_STATS.SET_SCHEMA_PREFS
  - DBMS_STATS.SET_DATABASE_PREFS
CF Improvement (TABLE_CACHED_BLOCKS)

```
SQL> exec dbms_stats.set_table_prefs(ownname=>user, tabname=>'BOWIE_ASSM',
                                               pname=>'TABLE_CACHED_BLOCKS', pvalue=>42);
PL/SQL procedure successfully completed.

SQL> exec dbms_stats.gather_index_stats(ownname=>user, indname=>'BOWIE_ASSM_ID_I',
                                                   estimate_percent=> null);
PL/SQL procedure successfully completed.

SQL> SELECT t.table_name, i.index_name, t.blocks, t.num_rows, i.clustering_factor
         FROM user_tables t, user_indexes i
         WHERE t.table_name = i.table_name AND i.index_name='BOWIE_ASSM_ID_I';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ASSM</td>
<td>BOWIE_ASSM_ID_I</td>
<td>1000</td>
<td>300000</td>
<td>909</td>
</tr>
</tbody>
</table>
```

Clustering Factor reduced from **217236** to just **909** !!
CF Improvement (TABLE_CACHED_BLOCKS)

SQL> select * from bowie_assm where id between 42 and 429;
388 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>389</td>
<td>6613</td>
<td>4 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>BOWIE_ASSM</td>
<td>389</td>
<td>6613</td>
<td>4 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BOWIE_ASSM_ID_I</td>
<td>389</td>
<td></td>
<td>2 (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
- 8734 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)
- 388 rows processed

Ensure No_Invalidate is set to false to ensure new plan generated after stats collection
CF Improvement (TABLE_CACHED_BLOCKS)

SQL> create table major_tom (id number, DOB date, text varchar2(42));
Table created.

SQL> insert into major_tom select rownum, sysdate-trunc(dbms_random.value(0, 20000)), 'DAVID BOWIE' from dual connect by level <= 2000000;
2000000 rows created.

SQL> commit;
Commit complete.

SQL> create index major_tom_dob_i on major_tom(dob);
Index created.

SQL> EXEC dbms_stats.gather_table_stats(ownname=>user, tabname=>'MAJOR_TOM', estimate_percent=>null, method_opt=>'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.

SQL> SELECT t.table_name, i.index_name, t.blocks, t.num_rows, i.clustering_factor
   FROM user_tables t, user_indexes i
   WHERE t.table_name = i.table_name AND i.index_name='MAJOR_TOM_DOB_I';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAJOR_TOM</td>
<td>MAJOR_TOM_DOB_I</td>
<td>9077</td>
<td>2000000</td>
<td>1988249</td>
</tr>
</tbody>
</table>
CF Improvement (TABLE_CACHED_BLOCKS)

```sql
SQL> exec dbms_stats.set_table_prefs(ownname=>user, tabname=>'MAJOR_TOM', 
    pname=>'TABLE_CACHED_BLOCKS', pvalue=>255);

PL/SQL procedure successfully completed.

SQL> EXEC dbms_stats.gather_index_stats(ownname=>user, indname=>'MAJOR_TOM_DOB_I', 
    estimate_percent=> null);

PL/SQL procedure successfully completed.

SQL> SELECT t.table_name, i.index_name, t.blocks, t.num_rows, i.clustering_factor 
  2  FROM user_tables t, user_indexes i 
  3  WHERE t.table_name = i.table_name AND i.index_name='MAJOR_TOM_DOB_I';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAJOR_TOM</td>
<td>MAJOR_TOM_DOB_I</td>
<td>9077</td>
<td>2000000</td>
<td>1941536</td>
</tr>
</tbody>
</table>
```

If the CF truly is poor, setting TABLE_CACHED_BLOCKS to the max of 255 quite correctly has little impact.
CF Improvement (TABLE_CACHED_BLOCKS)

SQL> select * from major_tom where dob between '01-JUN-2017' and '30-JUN-2017';

2955 rows selected. => Just 0.15% of data

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>3100</td>
<td>77500</td>
<td>2484 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>MAJOR_TOM</td>
<td>3100</td>
<td>77500</td>
<td>2484 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

- 0 recursive calls
- 0 db block gets
- 8582 consistent gets
- 0 physical reads
- 0 redo size
- 54489 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)
- 2955 rows processed
Warning With TABLE_CACHED_BLOCKS

SQL> alter index bowie_assm_id_i rebuild;
Index altered.

SQL> select * from bowie_assm where id between 42 and 429;
388 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>389</td>
<td>6613</td>
<td>282 (11)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>BOWIE_ASSM</td>
<td>389</td>
<td>6613</td>
<td>282 (11)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

3 recursive calls
0 db block gets
956 consistent gets
0 physical reads
0 redo size
4094 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)
388 rows processed
**Warning With TABLE_CACHED_BLOCKS**

```
SQL> SELECT t.table_name, i.index_name, t.blocks, t.num_rows, i.clustering_factor
FROM user_tables t, user_indexes i
WHERE t.table_name = i.table_name AND i.index_name='BOWIE_ASSM_ID_I';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ASSM</td>
<td>BOWIE_ASSM_ID_I</td>
<td>1000</td>
<td>300000</td>
<td>217236</td>
</tr>
</tbody>
</table>
```

SQL> exec dbms_stats.set_schema_prefs(ownname=>user, pname=>'TABLE_CACHED_BLOCKS', pvalue=>42);

PL/SQL procedure successfully completed.

SQL> exec dbms_stats.set_database_prefs(pname=>'TABLE_CACHED_BLOCKS', pvalue=>42);

PL/SQL procedure successfully completed.

SQL> alter index bowie_assm_id_i rebuild online;

Index altered.

```
SQL> SELECT t.table_name, i.index_name, t.blocks, t.num_rows, i.clustering_factor
FROM user_tables t, user_indexes i
WHERE t.table_name = i.table_name AND i.index_name='BOWIE_ASSM_ID_I';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ASSM</td>
<td>BOWIE_ASSM_ID_I</td>
<td>1000</td>
<td>300000</td>
<td>217236</td>
</tr>
</tbody>
</table>
```
Warning With TABLE_CACHED_BLOCKS

```
SQL> drop index bowie_assm_id_i;
Index dropped.

SQL> create index bowie_assm_id_i on bowie_assm(id);
Index created.

SQL> SELECT t.table_name, i.index_name, t.blocks, t.num_rows, i.clustering_factor
    FROM user_tables t, user_indexes i
    WHERE t.table_name = i.table_name AND i.index_name='BOWIE_ASSM_ID_I';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ASSM</td>
<td>BOWIE_ASSM_ID_I</td>
<td>1000</td>
<td>300000</td>
<td>217236</td>
</tr>
</tbody>
</table>
```
Table Re-Organisation To Maintain Performance
Online Table Move: Linear Order

How to improve performance of CODE based queries on the BOWIE table?

```sql
SQL> alter table bowie add clustering by linear order(code);
Table altered.

SQL> alter table bowie move online;
Table altered.

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>CLUSTERING_FACTOR</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ID_PK</td>
<td>7999955</td>
<td>8000000 (Previously 120986)</td>
</tr>
<tr>
<td>BOWIE_CODE_I</td>
<td>121030</td>
<td>8000000 (Previously 8000000)</td>
</tr>
<tr>
<td>BOWIE_GRADE_I</td>
<td>5875560</td>
<td>8000000 (Previously 5875580)</td>
</tr>
</tbody>
</table>
```

Clustering by just the CODE column improves its CF but has no impact on GRADE CF.
Online Table Move: Improve Performance

SQL> select * from bowie where code=42;

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>80000</td>
<td>8125K</td>
<td>1373 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BOWIE</td>
<td>80000</td>
<td>8125K</td>
<td>1373 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BOWIE_CODE_I</td>
<td>80000</td>
<td></td>
<td>159 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("CODE"=42)

Statistics

- 0 recursive calls
- 0 db block gets
- 1404 consistent gets (previously 80175 using index)
- 0 physical reads
- 0 redo size
- 8712544 bytes sent via SQL*Net to client
- 773 bytes received via SQL*Net from client
- 17 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 80000 rows processed
Query Based On GRADE Column

SQL> select * from bowie where grade=42;

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>80000</td>
<td>8125K</td>
<td>32913   (1)</td>
<td>00:00:02</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>BOWIE</td>
<td>80000</td>
<td>8125K</td>
<td>32913   (1)</td>
<td>00:00:02</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - filter("GRADE"=42)

Statistics

- 0 recursive calls
- 6 db block gets
- 121065 consistent gets
- 121032 physical reads
- 0 redo size
- 1908366 bytes sent via SQL*Net to client
- 783 bytes received via SQL*Net from client
- 18 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 80148 rows processed
Online Table Move: Interleaved Order

SQL> alter table bowie drop clustering;
Table altered.

SQL> alter table bowie add clustering by interleaved order (code, grade);
Table altered.

SQL> alter table bowie move online;
Table altered.

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>CLUSTERING_FACTOR</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE_ID_PK</td>
<td>7999859</td>
<td>8000000</td>
</tr>
<tr>
<td>BOWIE_CODE_I</td>
<td>130877</td>
<td>8000000 (Previously 121194 but much better than initial 8000000)</td>
</tr>
<tr>
<td>BOWIE_GRADE_I</td>
<td>126137</td>
<td>8000000 (Previously 5875560 so much improved)</td>
</tr>
</tbody>
</table>

Interleaved Clustering on both columns improves both CF (although the CF of CODE column is not quite as good compared to it clustered separately)
Online Table Move: Interleaved Order

SQL> select * from bowie where code=42;

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>80000</td>
<td>8125K</td>
<td>1470 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BOWIE</td>
<td>80000</td>
<td>8125K</td>
<td>1470 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BOWIE_CODE_I</td>
<td>80000</td>
<td></td>
<td>159 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("CODE"=42)

Statistics

0 recursive calls
0 db block gets
1504 consistent gets => A little more expensive than previous 1404 but still relatively efficient
0 physical reads
0 redo size
8712544 bytes sent via SQL*Net to client
772 bytes received via SQL*Net from client
17 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
80000 rows processed
Online Table Move: Interleaved Order

SQL> select * from bowie where grade=42;

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>80000</td>
<td>8125K</td>
<td>1423 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BOWIE</td>
<td>80000</td>
<td>8125K</td>
<td>1423 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BOWIE_GRADE_I</td>
<td>80000</td>
<td></td>
<td>159 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("GRADE"=42)

Statistics

0 recursive calls
0 db block gets
1450 consistent gets
0 physical reads
0 redo size
8727125 bytes sent via SQL*Net to client
783 bytes received via SQL*Net from client
18 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
80148 rows processed
Indexes and Sorts
Indexes and Sorts

- Indexes entries are always stored and sorted in the order of the indexed values (except for Reverse Key Indexes)
- Data retrieve via an Index Range Scan is always returned in index entry order
- Sorts are relatively expensive operations, especially if temp disk space is required
- Oracle can use the index as a method of retrieving data to avoid the sort operation
- But only if doing so is less costly than accessing all the necessary data via the associated index
- It's an incorrect assumption that an available index will always be used to avoid a sort ...
Indexes and Sorts – Demo Setup

**SQL**

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

SQL> CREATE SEQUENCE big_bowie_seq;
Sequence created.

SQL> create or replace procedure pop_big_bowie as
    begin
        for v_album_id in 1..10000 loop
            for v_country_id in 1..100 loop
                insert into big_bowie values (big_bowie_seq.nextval, v_album_id, ceil(dbms_random.value(0,100)), v_country_id, ceil(dbms_random.value(0,4)), trunc(sysdate-ceil(dbms_random.value(0,1000))), ceil(dbms_random.value(0,500000)));
            end loop;
        end loop;
    commit;
end;
/
Procedure created.

SQL> exec pop_big_bowie
PL/SQL procedure successfully completed.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BIG_BOWIE');
PL/SQL procedure successfully completed.
```
Indexes and Sorts – Demo Setup

```
SQL> CREATE TABLE big_bowie2 AS SELECT * FROM big_bowie_table 
    ORDER BY total_sales;
Table created.
SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'BIG_BOWIE2');
PL/SQL procedure successfully completed.
SQL> CREATE INDEX big_bowie_album_id_i ON big_bowie(album_id);
Index created.
SQL> CREATE INDEX big_bowie2_album_id_i ON big_bowie2(album_id);
Index created.
SQL> SELECT index_name, leaf_blocks, clustering_factor FROM user_indexes 
    WHERE index_name like 'BIG_BOWIE%ALBUM_ID_I';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>LEAF_BLOCKS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG_BOWIE2_ALBUM_ID_I</td>
<td>2090</td>
<td>989911</td>
</tr>
<tr>
<td>BIG_BOWIE_ALBUM_ID_I</td>
<td>2090</td>
<td>5925</td>
</tr>
</tbody>
</table>
```
Indexes and Sorts

SQL> SELECT * FROM big_bowie WHERE album_id BETWEEN 10 AND 20 ORDER BY id;
1100 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>1200</td>
<td>37200</td>
<td>15 (14)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>SORT ORDER BY</td>
<td></td>
<td>1200</td>
<td>37200</td>
<td>15 (14)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>BIG_BOWIE</td>
<td>1200</td>
<td>37200</td>
<td>13 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 3</td>
<td>INDEX RANGE SCAN</td>
<td>BIG_BOWIE_ALBUM_ID_I</td>
<td>1200</td>
<td></td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
11 consistent gets
0 physical reads
0 redo size
48222 bytes sent via SQL*Net to client
1355 bytes received via SQL*Net from client
75 SQL*Net roundtrips to/from client
1 sorts (memory)
0 sorts (disk)
1100 rows processed
Indexes and Sorts

SQL> SELECT * FROM big_bowie WHERE album_id BETWEEN 10 AND 20 ORDER BY album_id;
1100 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>1200</td>
<td>37200</td>
<td>13 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_BOWIE</td>
<td>1200</td>
<td>37200</td>
<td>13 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BIG_BOWIE_ALBUM_ID_I</td>
<td>1200</td>
<td></td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

- 0 recursive calls
- 0 db block gets
- 13 consistent gets
- 0 physical reads
- 0 redo size
- 38364 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)
- 1100 rows processed
Indexes and Sorts

SQL> SELECT * FROM big_bowie WHERE album_id BETWEEN 1 AND 10000 ORDER BY album_id;
1000000 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>1000K</td>
<td>29M</td>
<td>8252 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_BOWIE</td>
<td>1000K</td>
<td>29M</td>
<td>8252 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*  2</td>
<td>INDEX RANGE SCAN</td>
<td>BIG_BOWIE_ALBUM_ID_I</td>
<td>1000K</td>
<td></td>
<td>2188 (5)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
139919 consistent gets
0 physical reads
0 redo size
37476458 bytes sent via SQL*Net to client
733722 bytes received via SQL*Net from client
66668 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1000000 rows processed

Index retrieves **entire table** as doing so is cheaper than performing the necessary sort ...
Indexes and Sorts

SQL> SELECT * FROM big_bowie ORDER BY album_id;
1000000 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>TempSpc</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1000K</td>
<td>29M</td>
<td></td>
<td>10779 (6)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>SORT ORDER BY</td>
<td></td>
<td>1000K</td>
<td>29M</td>
<td>49M</td>
<td>10779 (6)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS FULL</td>
<td>BIG_BOWIE</td>
<td>1000K</td>
<td>29M</td>
<td></td>
<td>1397 (10)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

- 0 recursive calls
- 0 db block gets
- 4983 consistent gets
- 0 physical reads
- 0 redo size
- 43937515 bytes sent via SQL*Net to client
- 733878 bytes received via SQL*Net from client
- 66668 SQL*Net roundtrips to/from client
- 1 sorts (memory)
- 0 sorts (disk)
- 1000000 rows processed

So why isn't the index being used now??
Indexes and Sorts

SQL> SELECT * FROM big_bowie WHERE album_id IS NOT NULL ORDER BY album_id;
1000000 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>1000K</td>
<td>29M</td>
<td>8261 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_BOWIE</td>
<td>1000K</td>
<td>29M</td>
<td>8261 (3)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX FULL SCAN</td>
<td>BIG_BOWIE_ALBUM_ID_I</td>
<td>1000K</td>
<td></td>
<td>2188 (5)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
139919 consistent gets
7040 physical reads
0 redo size
37476493 bytes sent via SQL*Net to client
733722 bytes received via SQL*Net from client
66668 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1000000 rows processed

By including IS NOT NULL, can guarantee all required rows are indexed and can use index to avoid the sort. The cost of 8261 is better than the cost of 10779 for the corresponding FTS.
Indexes and Sorts

SQL> SELECT * FROM big_bowie2 WHERE album_id IS NOT NULL ORDER BY album_id;
1000000 rows selected.

Execution Plan

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>TempSpc</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1000K</td>
<td>29M</td>
<td></td>
<td>10795 (6)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>SORT ORDER BY</td>
<td></td>
<td>1000K</td>
<td>29M</td>
<td>49M</td>
<td>10795 (6)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>*  2</td>
<td>TABLE ACCESS FULL</td>
<td>BIG_BOWIE2</td>
<td>1000K</td>
<td>29M</td>
<td></td>
<td>1413 (11)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

- 0 recursive calls
- 0 db block gets
- 4981 consistent gets
- 0 physical reads
- 0 redo size
- 43934622 bytes sent via SQL*Net to client
- 733878 bytes received via SQL*Net from client
- 66668 SQL*Net roundtrips to/from client
- 1 sorts (memory)
- 0 sorts (disk)
- 1000000 rows processed

The situation is dramatically different for BIG_BOWIE2 which has a terrible clustering factor. Oracle determines the sort is cheaper than reading the table via the inefficient index.
Indexes and Sorts

SQL> SELECT /*+ index(a) */ * from big_bowie2 a WHERE album_id IS NOT NULL ORDER BY album_id;
1000000 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>1000K</td>
<td>29M</td>
<td>995K (1)</td>
<td>00:00:39</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_BOWIE2</td>
<td>1000K</td>
<td>29M</td>
<td>995K (1)</td>
<td>00:00:39</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX FULL SCAN</td>
<td>BIG_BOWIE2_ALBUM_ID_I</td>
<td>1000K</td>
<td></td>
<td>2188 (5)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
1059209 consistent gets
0 physical reads
0 redo size
48863561 bytes sent via SQL*Net to client
733878 bytes received via SQL*Net from client
66668 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1000000 rows processed

Note: Forcing the Index Full Scan via a hint highlights the cost at a massive 995K ...
Indexes and Sorts

SQL> SELECT * from big_bowie2 WHERE album_id BETWEEN 1 AND 15 ORDER BY album_id;
1500 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>1500</td>
<td>46500</td>
<td>1383</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>SORT ORDER BY</td>
<td></td>
<td>1500</td>
<td>46500</td>
<td>1383</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>BIG_BOWIE2</td>
<td>1500</td>
<td>46500</td>
<td>1381</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
4981 consistent gets
0 physical reads
0 redo size
54139 bytes sent via SQL*Net to client
1485 bytes received via SQL*Net from client
101 SQL*Net roundtrips to/from client
1 sorts (memory)
0 sorts (disk)
1500 rows processed

Note: Even with selectivity as low as 0.15%, a FTS and sort is still preferred over an index scan
Indexes and Sorts

SQL> SELECT * from big_bowie2 WHERE album_id BETWEEN 1 AND 13 ORDER BY album_id;
1300 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>1300</td>
<td>40300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_BOWIE2</td>
<td>1300</td>
<td>40300</td>
<td>1297 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BIG_BOWIE2_ALBUM_ID_I</td>
<td>1300</td>
<td></td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Statistics

0 recursive calls
0 db block gets
1381 consistent gets
0 physical reads
0 redo size
61737 bytes sent via SQL*Net to client
1498 bytes received via SQL*Net from client
88 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1300 rows processed

In this scenario, it wasn't until 0.13% selectivity the index access cost of 1297 was finally cheaper than FTS and sort cost of 1383. The CF is indeed critical ...
Partitioning To Improve Performance
Index Partitioning Options

Global Non-Partitioned

Global Partitioned

Local Partitioned

Partitioned Table

Non-Partitioned Table
Online Conversion To Partitioned Table

SQL> alter table bowie
2  modify partition by range (code)
3       (partition p0 values less than (1),
4       partition p1 values less than (2),
5       partition p2 values less than (3),
6       partition p3 values less than (4),
7       partition p4 values less than (5),
8       partition p5 values less than (6),
9       partition p6 values less than (7),
10      partition p7 values less than (8),
11      partition p8 values less than (9),
12      partition p9 values less than (10),
...
100     partition p97 values less than (97),
101     partition p98 values less than (98),
102     partition p99 values less than (maxvalue)) online;

Table altered.
Online Conversion To Partitioned Table

SQL> select table_name, status, partitioned from user_tables where table_name='Bowie';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>STATUS</th>
<th>PAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE</td>
<td>VALID</td>
<td>YES</td>
</tr>
</tbody>
</table>

SQL> select table_name, partition_name, num_rows from user_tab_partitions where table_name='BOWIE';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>PARTITION_NAME</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOWIE</td>
<td>P66</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P97</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P98</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P99</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P43</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P44</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P45</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P46</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P47</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P48</td>
<td>80000</td>
</tr>
<tr>
<td>BOWIE</td>
<td>P18</td>
<td>80000</td>
</tr>
</tbody>
</table>
...
Partition Pruning: Partition-Level Scan

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

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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>80000</td>
<td>8125K</td>
<td>340 (1)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PARTITION RANGE SINGLE</td>
<td></td>
<td>80000</td>
<td>8125K</td>
<td>340 (1)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>TABLE ACCESS FULL</td>
<td>BOWIE</td>
<td>80000</td>
<td>8125K</td>
<td>340 (1)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - filter("CODE"=42)

Statistics

0 recursive calls
5 db block gets
1244 consistent gets ➔ More efficient than index scan
0 physical reads
0 redo size
1905680 bytes sent via SQL*Net to client
773 bytes received via SQL*Net from client
17 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
80000 rows processed
```
Index On CODE Column Now Redundant

Partitioning Pruning is now more efficient than corresponding index as all required data and nothing but required data can be retrieved via a partition-level full scan, using efficient multi-block reads...

```
SQL> drop index bowie_code_i;
Index dropped.
```
Still Effectively Use Index GRADE Column

SQL> select * from bowie where grade=42;

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>80148</td>
<td>8140K</td>
<td>1859 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY GLOBAL INDEX ROWID</td>
<td>BOWIE</td>
<td>80148</td>
<td>8140K</td>
<td>1859 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BOWIE_GRADE_I</td>
<td>82657</td>
<td></td>
<td>211 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

   2  - access("GRADE"=42)

Statistics

  0  recursive calls
  0  db block gets
  1635  consistent gets
  0  physical reads
  0  redo size
8727125  bytes sent via SQL*Net to client
  784  bytes received via SQL*Net from client
  18  SQL*Net roundtrips to/from client
  0  sorts (memory)
  0  sorts (disk)
80148  rows processed
Partition To Reduce Maintenance Overheads
Remember The ZIGGY Table

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>CLUSTERING_FACTOR</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZIGGY_GRADE_I</td>
<td>5874495</td>
<td>8000000</td>
</tr>
<tr>
<td>ZIGGY_ID_PK</td>
<td>7999951</td>
<td>8000000</td>
</tr>
<tr>
<td>ZIGGY_CODE_I</td>
<td>121032</td>
<td>8000000</td>
</tr>
</tbody>
</table>

SQL> alter table ziggy add clustering by interleaved order (code, grade);

Table altered.

SQL> alter table ziggy move online;

Table altered.

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>CLUSTERING_FACTOR</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZIGGY_GRADE_I</td>
<td>126117</td>
<td>8000000</td>
</tr>
<tr>
<td>ZIGGY_ID_PK</td>
<td>7999934</td>
<td>8000000</td>
</tr>
<tr>
<td>ZIGGY_CODE_I</td>
<td>130849</td>
<td>8000000</td>
</tr>
</tbody>
</table>
Current Query on Code Column: All Good

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

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>80000</td>
<td>8125K</td>
<td>1470      (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ZIGGY</td>
<td>80000</td>
<td>8125K</td>
<td>1470      (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>ZIGGY_CODE_I</td>
<td>80000</td>
<td></td>
<td>159       (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("CODE"=42)

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>1505</td>
</tr>
<tr>
<td>physical reads</td>
<td>0</td>
</tr>
<tr>
<td>redo size</td>
<td>0</td>
</tr>
<tr>
<td>bytes sent via SQL*Net to client</td>
<td>8712544</td>
</tr>
<tr>
<td>bytes received via SQL*Net from client</td>
<td>773</td>
</tr>
<tr>
<td>SQL*Net roundtrips to/from client</td>
<td>17</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>80000</td>
</tr>
</tbody>
</table>
Add Some More Rows To ZIGGY Table...

```
SQL> insert into ziggy select rownum+8000000, mod(rownum,100), ceil(dbms_random.value(0,100)),
          ceil(dbms_random.value(0,10000)), 'PROCESSED', sysdate+mod(rownum,30), 'DAVID BOWIE', 'ZIGGY STARDUST', 'THE THIN WHITE DUKE', 'ALADDIN SANE', 'MAJOR TOM' from dual connect by level <=250000;
250000 rows created.

SQL> commit;
Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'ZIGGY', estimate_percent=>100, method_opt=>'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>CLUSTERING_FACTOR</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZIGGY_GRADE_I</td>
<td>309526</td>
<td>8250000 (Previously 126117)</td>
</tr>
<tr>
<td>ZIGGY_ID_PK</td>
<td>8003722</td>
<td>8250000</td>
</tr>
<tr>
<td>ZIGGY_CODE_I</td>
<td>380849</td>
<td>8250000 (Previously 130849)</td>
</tr>
</tbody>
</table>
```
Impact Of New Rows To Performance

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

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>82500</td>
<td>8378K</td>
<td>3979        (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>ZIGGY</td>
<td>82500</td>
<td>8378K</td>
<td>3979        (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>ZIGGY_CODE_I</td>
<td>82500</td>
<td></td>
<td>167        (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("CODE"=42)

Statistics

0 recursive calls
0 db block gets
4015 consistent gets (Previously 1505, 167% increase)
0 physical reads
0 redo size
8985194 bytes sent via SQL*Net to client
784 bytes received via SQL*Net from client
18 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
82500 rows processed (Previously 80000, 3% increase)
```
Partition To Avoid Full Table/Index Rebuilds

SQL> alter table ziggy
    modify partition by range (date_entered)
    (partition p1 values less than (TO_DATE('01-JAN-2014', 'DD-MON-YYYY'))),
    partition p2 values less than (TO_DATE('01-JAN-2015', 'DD-MON-YYYY'))),
    partition p3 values less than (TO_DATE('01-JAN-2016', 'DD-MON-YYYY'))),
    partition p4 values less than (TO_DATE('01-JAN-2017', 'DD-MON-YYYY'))),
    partition p5 values less than (TO_DATE('01-JAN-2018', 'DD-MON-YYYY'))),
    partition p6 values less than (maxvalue) online
    update indexes (ziggy_code_i local);

Table altered.

SQL> alter table ziggy move partition p6 update indexes online;

Table altered.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'ZIGGY',
estimate_percent=>100, method_opt=>'FOR ALL COLUMNS SIZE 1');

PL/SQL procedure successfully completed.
Partition to Avoid Full Table/Index Rebuilds

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

<table>
<thead>
<tr>
<th>PARTITION_NAME</th>
<th>NUM_ROWS</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1178758</td>
<td>27528</td>
</tr>
<tr>
<td>P2</td>
<td>1599900</td>
<td>33899</td>
</tr>
<tr>
<td>P3</td>
<td>1600160</td>
<td>33922</td>
</tr>
<tr>
<td>P4</td>
<td>1604544</td>
<td>33988</td>
</tr>
<tr>
<td>P5</td>
<td>1600160</td>
<td>33905</td>
</tr>
<tr>
<td>P6</td>
<td>666478</td>
<td>19576</td>
</tr>
</tbody>
</table>
```

```sql
SQL> select index_name, clustering_factor, num_rows from user_indexes
   where table_name='ZIGGY';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>CLUSTERING_FACTOR</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZIGGY_ID_PK</td>
<td>8243694</td>
<td>8250000</td>
</tr>
<tr>
<td>ZIGGY_CODE_I</td>
<td>182818</td>
<td>8250000</td>
</tr>
<tr>
<td>ZIGGY_GRADE_I</td>
<td>154613</td>
<td>8250000</td>
</tr>
</tbody>
</table>
```
Partition to Avoid Full Table/Index Rebuilds

SQL> select * from ziggy where code=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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>82500</td>
<td>8378K</td>
<td>2005</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>82500</td>
<td>8378K</td>
<td>2005</td>
<td>00:00:01</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY LOCAL INDEX ROWID</td>
<td>ZIGGY</td>
<td>82500</td>
<td>8378K</td>
<td>2005</td>
<td>00:00:01</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>* 3</td>
<td>INDEX RANGE SCAN</td>
<td>ZIGGY_CODE_I</td>
<td>82500</td>
<td></td>
<td>174</td>
<td>00:00:01</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
3 - access("CODE"=42)

Statistics

- 0 recursive calls
- 0 db block gets
- 2053 consistent gets (Down from 4015)
- 0 physical reads
- 132 redo size
- 1922040 bytes sent via SQL*Net to client
- 784 bytes received via SQL*Net from client
- 18 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 82500 rows processed
Partial Indexes
Partial Indexes (12c)

SQL> create table pink_floyd (id number, status varchar2(6), name varchar2(42))
    2    indexing off
    3    partition by range (id)
    4    (partition pf1 values less than (1000001),
    5    partition pf2 values less than (2000001) indexing off,
    6    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> commit;
Commit complete.

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

SQL> commit;
Commit complete.

New INDEXING partitioned table option allows data from specific partitions to be excluded from indexes
Partial Indexes

```
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';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>NUM_ROWS</th>
<th>LEAF_BLOCKS</th>
<th>INDEXING</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINK_FLOYD_STATUS_I</td>
<td>3000000</td>
<td>9203</td>
<td>FULL</td>
</tr>
</tbody>
</table>
```

Default is for an index to be a “FULL” index
Partial Indexes

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

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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>99</td>
<td>2475</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED</td>
<td>PINK_FLOYD</td>
<td>99</td>
<td>2475</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>PINK_FLOYD_STATUS_I</td>
<td>100</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>100</td>
<td>2500</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED</td>
<td>PINK_FLOYD</td>
<td>100</td>
<td>2500</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td>ROWID</td>
<td>ROWID</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>PINK_FLOYD_STATUS_I</td>
<td>100</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Partial Indexes

SQL> drop index pink_floyd_status_i;
Index dropped.

SQL> create index pink_floyd_status_i on pink_floyd(status) indexing partial;
Index created.

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>NUM_ROWS</th>
<th>LEAF_BLOCKS</th>
<th>INDEXING</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINK_FLOYD_STATUS_I</td>
<td>1000000</td>
<td>3068</td>
<td>PARTIAL</td>
</tr>
</tbody>
</table>

Only those partitions with “INDEXING ON” are included within the index. Therefore index is approx. 1/3 the size.
Partial Indexes

SQL> select * from pink_floyd where status = 'OPEN' and id >= 2000001;
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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>100</td>
<td>2500</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED</td>
<td>PINK_FLOYD</td>
<td>100</td>
<td>2500</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>PINK_FLOYD_STATUS_I</td>
<td>33</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If queries only access “active” partitions, then the partial index is sufficient
Partial Indexes

```sql
SQL> select * from pink_floyd where status = 'OPEN';
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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>100</td>
<td>2500</td>
<td>2491 (9)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>VIEW</td>
<td>VW_TE_2</td>
<td>99</td>
<td>4059</td>
<td>2491 (9)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UNION-ALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED</td>
<td>PINK_FLOYD</td>
<td>33</td>
<td>825</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td>ROWID</td>
<td>ROWID</td>
</tr>
<tr>
<td>* 4</td>
<td>INDEX RANGE SCAN</td>
<td>PINK_FLOYD_STATUS_I</td>
<td>100</td>
<td>825</td>
<td>4 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PARTITION RANGE ITERATOR</td>
<td></td>
<td>66</td>
<td>1650</td>
<td>2487 (9)</td>
<td>00:00:01</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>* 6</td>
<td>TABLE ACCESS FULL</td>
<td>PINK_FLOYD</td>
<td>66</td>
<td>1650</td>
<td>2487 (9)</td>
<td>00:00:01</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
```

If queries sometimes access “inactive” partitions, then a combination of partial indexes and full partition scans might be required
Local Partial Indexes

SQL> drop index pink_floyd_status_i;
Index dropped.

SQL> create index pink_floyd_status_i on pink_floyd(status) local indexing partial;
Index created.

SQL> select index_name, partition_name, num_rows, status, leaf_blocks from user_ind_partitions
where index_name = 'PINK_FLOYD_STATUS_I';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>PARTITION_NAME</th>
<th>NUM_ROWS</th>
<th>STATUS</th>
<th>LEAF_BLOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINK_FLOYD_STATUS_I</td>
<td>PK1</td>
<td>0</td>
<td>UNUSABLE</td>
<td>0</td>
</tr>
<tr>
<td>PINK_FLOYD_STATUS_I</td>
<td>PK2</td>
<td>0</td>
<td>UNUSABLE</td>
<td>0</td>
</tr>
<tr>
<td>PINK_FLOYD_STATUS_I</td>
<td>PK3</td>
<td>1000000</td>
<td>USABLE</td>
<td>2507</td>
</tr>
</tbody>
</table>

If partial index is Local, then inactive partitions are marked as unusable and consume no storage...
Partial Indexes

To avoid possible FTS of inactive partitions, ensure data of interest can only reside in indexing partitions. In 18c, can modify a partitioned table to be partitioned entirely differently, online...

```sql
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)) enable row movement online;
Table altered.
```

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

<table>
<thead>
<tr>
<th>SUBPARTITION_POSITION</th>
<th>SUBPARTITION_NAME</th>
<th>NUM_ROWS</th>
<th>INDEXING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PF1_CLOSED</td>
<td>1000000</td>
<td>OFF</td>
</tr>
<tr>
<td>2</td>
<td>PF1_OPEN</td>
<td>0</td>
<td>ON</td>
</tr>
<tr>
<td>1</td>
<td>PF2_CLOSED</td>
<td>1000000</td>
<td>OFF</td>
</tr>
<tr>
<td>2</td>
<td>PF2_OPEN</td>
<td>0</td>
<td>ON</td>
</tr>
<tr>
<td>1</td>
<td>PF3_CLOSED</td>
<td>999900</td>
<td>OFF</td>
</tr>
<tr>
<td>2</td>
<td>PF3_OPEN</td>
<td>100</td>
<td>ON</td>
</tr>
</tbody>
</table>
Partial Indexes

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>NUM_ROWS</th>
<th>LEAF_BLOCKS</th>
<th>INDEXING</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINK_FLOYD_STATUS_I</td>
<td>100</td>
<td>1</td>
<td>PARTIAL</td>
</tr>
</tbody>
</table>

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

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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>100</td>
<td>2500</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED</td>
<td>PINK_FLOYD</td>
<td>100</td>
<td>2500</td>
<td>2 (0)</td>
<td>00:00:01</td>
<td>ROWID</td>
<td>ROWID</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>PINK_FLOYD_STATUS_I</td>
<td>100</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
- 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
Partition Definition and Data Integrity

SQL> insert into ziggy2 select 8000001, 42, 42, 42, 'PROCESSED', sysdate, 'DAVID BOWIE', 'ZIGGY STARDUST', 'THE THIN WHITE DUKE', 'ALADDIN SANE', 'MAJOR TOM' from dual;

insert into ziggy2 select 8000001, 42, 42, 42, 'CLOSED', sysdate, 'DAVID BOWIE', 'ZIGGY STARDUST', 'THE THIN WHITE DUKE', 'ALADDIN SANE', 'MAJOR TOM' from dual
  *
ERROR at line 1:
ORA-14400: inserted partition key does not map to any partition
Partition Definition and Data Integrity

SQL> select * from ziggy2 where status = 'PROCESSED';

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>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>104</td>
<td>34284   (1)</td>
<td>00:00:02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>1</td>
<td>104</td>
<td>34284   (1)</td>
<td>00:00:02</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>PARTITION LIST EMPTY</td>
<td></td>
<td>1</td>
<td>104</td>
<td>34284   (1)</td>
<td>00:00:02</td>
<td>INVALID</td>
<td>INVALID</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS FULL</td>
<td>ZIGGY2</td>
<td>1</td>
<td>104</td>
<td>34284   (1)</td>
<td>00:00:02</td>
<td>INVALID</td>
<td>INVALID</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

3 - filter("STATUS"='CLOSED')

Statistics

0 recursive calls
0 db block gets
0 consistent gets
0 physical reads
0 redo size
1015 bytes sent via SQL*Net to client
597 bytes received via SQL*Net from client
1 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
0 rows processed
“Manual” Partial Indexes

SQL> drop index ziggy_status_i;

Index dropped.

SQL> CREATE INDEX ziggy_status_i ON ziggy(DECODE(status, 'OPEN', 'OPEN', NULL));

Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'ZIGGY', estimate_percent=>100, method_opt=> 'FOR ALL HIDDEN COLUMNS SIZE 1');

PL/SQL procedure successfully completed.

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

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>NUM_ROWS</th>
<th>LEAF_BLOCKS</th>
<th>BLEVEL</th>
<th>INDEXING</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZIGGY_STATUS_I</td>
<td>8333</td>
<td>19</td>
<td>1</td>
<td>FULL</td>
</tr>
</tbody>
</table>
“Manual” Partial Indexes

SQL> select * from ziggy where decode(status, 'OPEN', 'OPEN', NULL) = 'OPEN';

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>8333</td>
<td>854K</td>
<td>1653 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY GLOBAL INDEX ROWID</td>
<td>ZIGGY</td>
<td>8333</td>
<td>854K</td>
<td>1653 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>ZIGGY_STATUS_I</td>
<td>8333</td>
<td></td>
<td>24 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access(DECODE("STATUS","OPEN","OPEN",NULL)='OPEN')

Statistics

0 recursive calls
0 db block gets
1657 consistent gets
0 physical reads
0 redo size
137242 bytes sent via SQL*Net to client
619 bytes received via SQL*Net from client
3 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
8333 rows processed
A Query That Appears To Be Fine

SQL> create table major_tom (id number, code number, name varchar2(42));
Table created.

SQL> insert into major_tom select rownum, mod(rownum,2000)+1, 'DAVID BOWIE'
    from dual connect by level <=2000000;
2000000 rows created.

SQL> commit;
Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'MAJOR_TOM');
PL/SQL procedure successfully completed.

SQL> create index major_tom_code_i on major_tom(code);
Index created.
Query Appears Fine – What’s “Wrong”?

SQL> select * from major_tom where code=42;
Elapsed: 00:00:00.00

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>1000</td>
<td></td>
<td>0</td>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>MAJOR_TOM</td>
<td>1000</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>MAJOR_TOM_CODE_I</td>
<td>1000</td>
<td></td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("CODE"=42)

Statistics

0 recursive calls
0 db block gets
1006 consistent gets
0 physical reads
0 redo size
26208 bytes sent via SQL*Net to client
607 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1000 rows processed
A Query That Appears To Be Fine

```sql
SQL> select index_name, clustering_factor from user_indexes where table_name='MAJOR_TOM';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAJOR_TOM_CODE_I</td>
<td>2000000</td>
</tr>
</tbody>
</table>

SQL> alter table major_tom add clustering by linear order(code);

Table altered.

SQL> alter table major_tom move online;

Table altered.

SQL> select index_name, clustering_factor from user_indexes where table_name='MAJOR_TOM';

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>CLUSTERING_FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAJOR_TOM_CODE_I</td>
<td>7322</td>
</tr>
</tbody>
</table>
```
A Query That Appears To Be Fine

SQL> select * from major_tom where code=42;
Elapsed: 00:00:00.00

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 BY INDEX ROWID</td>
<td>MAJOR_TOM</td>
<td>1000</td>
<td>21000</td>
<td>9 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>MAJOR_TOM_CODE_I</td>
<td>1000</td>
<td></td>
<td>5 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("CODE"=42)

Statistics

0 recursive calls
0 db block gets
12 consistent gets (Approx. just 1% of the previous CPU costs)
0 physical reads
0 redo size
26208 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)
1000 rows processed
The Danger of Outlier Values
How Just One New Row Can Cause Havoc...

SQL> select * from ziggy where code>=99;
Elapsed: 00:00:00.02

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>80000</td>
<td>1373   (1)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>ZIGGY</td>
<td>80000</td>
<td>8125K</td>
<td>1373 (1)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>ZIGGY_CODE_I</td>
<td>80000</td>
<td></td>
<td>159 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

2 - access("CODE">=99)

Statistics

- 0 recursive calls
- 0 db block gets
- 1404 consistent gets
- 0 physical reads
- 0 redo size

- 8712465 bytes sent via SQL*Net to client
- 773 bytes received via SQL*Net from client
- 17 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 80000 rows processed
How Just One New Row Can Cause Havoc...

SQL> insert into ziggy select 8250001, 9999999999, 42, 42, 'CLOSED', sysdate, 'DAVID BOWIE', 'ZIGGY STARDUST', 'THE THIN WHITE DUKE', 'ALADDIN SANE', 'MAJOR TOM' from dual;

1 row created.

SQL> commit;

Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'ZIGGY',estimate_percent=>100, method_opt=> 'FOR ALL COLUMNS SIZE 1');

PL/SQL procedure successfully completed.

SQL> select min(code), max(code) from ziggy;

<table>
<thead>
<tr>
<th>MIN(CODE)</th>
<th>MAX(CODE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0000000000</td>
<td>9999999999</td>
</tr>
</tbody>
</table>
How Just One New Row Can Cause Havoc...

SQL> select * from ziggy where code>=99;
Elapsed: 00:00:09.71

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>8000K</td>
<td>793M</td>
<td>33201</td>
<td>0:00:02</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS FULL</td>
<td>ZIGGY</td>
<td>8000K</td>
<td>793M</td>
<td>33201</td>
<td>0:00:02</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - filter("CODE">=99)

Statistics

0 recursive calls
0 db block gets
121225 consistent gets
121200 physical reads
0 redo size
1905381 bytes sent via SQL*Net to client
784 bytes received via SQL*Net from client
18 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
80001 rows processed
Accessing NULL Values
Outliers Often Used To Avoid NULL Values

```sql
SQL> alter table ziggy modify code null;
Table altered.

SQL> update ziggy set code=null where id= 8250001;
1 row updated.

SQL> commit;
Commit complete.

SQL> exec dbms_stats.gather_table_stats(ownname=>null, tabname=>'ZIGGY', estimate_percent=>100, method_opt=> 'FOR ALL COLUMNS SIZE 1');
PL/SQL procedure successfully completed.
```
Outliers Often Used To Avoid NULL Values

SQL> select * from ziggy where code>=99;  ===> All good again

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></td>
<td>SELECT STATEMENT</td>
<td></td>
<td>80000</td>
<td>8125K</td>
<td>1375</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>ZIGGY</td>
<td>80000</td>
<td>8125K</td>
<td>1375</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>ZIGGY_CODE_I</td>
<td>80000</td>
<td></td>
<td>161</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):
2 - access("CODE">=99)

Statistics

0 recursive calls
0 db block gets
12047 consistent gets
0 physical reads
0 redo size
9744157 bytes sent via SQL*Net to client
59271 bytes received via SQL*Net from client
5335 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
80000 rows processed
Entirely NULL Column Values Not Indexed

```
SQL> select * from ziggy where code is null;

Execution Plan
---------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |     1 |   104 | 34226   (1)| 00:00:02 |
|*  1 | TABLE ACCESS FULL| ZIGGY |     1 |   104 | 34226   (1)| 00:00:02 |
---------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
1  - filter("CODE" IS NULL)

Statistics
----------------------------------------------------------
0  recursive calls
0  db block gets
125114  consistent gets
125101  physical reads
0  redo size
1311  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
```
NULLs Indexed If Other Indexed Value Is Not NULL

SQL> create index ziggy_code_i2 on ziggy(code, ' ');
Index created.

Above CODE nullable column is concatenated with a constant (a space) which can’t be NULL. Therefore all NULL code values are guaranteed to be indexed.
Index Now Used To Access NULL Values

```
SQL> select * from ziggy where code is null;

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>104</td>
<td>4 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>ZIGGY</td>
<td>1</td>
<td>104</td>
<td>4 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>ZIGGY_CODE_I2</td>
<td>1</td>
<td></td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

  2 - access("CODE" IS NULL)

Statistics

  0 recursive calls
  0 db block gets
  4 consistent gets
  0 physical reads
  0 redo size
  1311 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
```
Clustering Data on Exadata Storage
Storage Indexes

Select * from Bowie where B=2
Storage Indexes – Demo Setup

SQL> create table big_bowie (id number not null, album_id number not null, artist_id number not null, format_id number, release_date date, total_sales number, description varchar2(100));
Table created.

SQL> create sequence bowie_seq order;
Sequence created.

SQL> create or replace procedure pop_big_bowie as
2  begin
3     for v_album_id in 1..100 loop
4         for v_artist_id in 1..100000 loop
5             insert into big_bowie values (bowie_seq.nextval, v_album_id, v_artist_id, ceil(dbms_random.value(0,5)) * 2, trunc(sysdate-ceil(dbms_random.value(0,10000))), ceil(dbms_random.value(0,500000)), 'THE RISE AND FALL OF ZIGGY STARDUST AND THE SPIDERS FROM MARS');
6         end loop;
7     end loop;
8     commit;
9 end;
10 /
Procedure created.

SQL> exec pop_big_bowie
PL/SQL procedure successfully completed.
Storage Indexes – “Clustering Factor”

ALBUM_ID column data is well clustered ...

```
SQL> select * from big_bowie where album_id = 42;
100000 rows selected.
Elapsed: 00:00:00.27

Execution Plan
---------------------------------------------------------------------------------------
| Id  | Operation                 | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |           |   100K|  8984K| 36663   (1)| 00:07:20 |
|*  1 |  TABLE ACCESS STORAGE FULL| BIG_BOWIE |   100K|  8984K| 36663   (1)| 00:07:20 |
---------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
1 - storage("ALBUM_ID"=42)
   filter("ALBUM_ID"=42)
```

Statistics
```
1 recursive calls
0 db block gets
134834 consistent gets
134809 physical reads
0 redo size
4345496 bytes sent via SQL*Net to client
73850 bytes received via SQL*Net from client
6668 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
100000 rows processed
```
Storage Indexes – “Clustering Factor”

```sql
SQL> select name, value/1024/1024 MB from v$statname n, v$mystat s where n.statistic# = s.statistic# and n.name in ('cell physical IO interconnect bytes returned by smart scan', 'cell physical IO bytes saved by storage index');
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell physical IO bytes saved by storage index</td>
<td>1042.24219</td>
</tr>
<tr>
<td>cell physical IO interconnect bytes returned by smart scan</td>
<td>9.56161499</td>
</tr>
</tbody>
</table>

The Storage Index has managed to avoid having to read approx. 99% of the table.

Having data well clustered can result in Partition-Pruning like efficiencies when using an associated Storage Index …
Storage Indexes – “Clustering Factor”

Even when all the data is cached ...

```
SQL> select * from big_bowie where album_id = 42;
100000 rows selected.
Elapsed: 00:00:00.27

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>8984K</td>
<td>1550 (1)</td>
<td>00:00:19</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>BIG_BOWIE</td>
<td>100K</td>
<td>8984K</td>
<td>1550 (1)</td>
<td>00:00:19</td>
</tr>
<tr>
<td>* 2</td>
<td>INDEX RANGE SCAN</td>
<td>BIG_BOWIE_ALBUM_ID_I</td>
<td>100K</td>
<td>199</td>
<td>1550 (1)</td>
<td>00:00:03</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

- 2 - access("ALBUM_ID"=42)

Statistics

0  recursive calls
0  db block gets
1590  consistent gets
0  physical reads
0  redo size
9689267  bytes sent via SQL*Net to client
733  bytes received via SQL*Net from client
21  SQL*Net roundtrips to/from client
0  sorts (memory)
0  sorts (disk)
100000  rows processed
```

Index is no more efficient than FTS, even though the index is extremely efficient ...
Storage Indexes – “Clustering Factor”

The TOTAL_SALES column however is not well clustered ...

SQL> select album_id, artist_id from big_bowie where total_sales between 42 and 142;
2009 rows selected.
Elapsed: 00:00:01.25 => 5 x Slower

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>2040</td>
<td>26520</td>
<td>36700   (1)</td>
<td>00:07:21</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS STORAGE FULL</td>
<td>BIG_BOWIE</td>
<td>2040</td>
<td>26520</td>
<td>36700   (1)</td>
<td>00:07:21</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - storage("TOTAL_SALES"<=142 AND "TOTAL_SALES">=42)
  filter("TOTAL_SALES"<=142 AND "TOTAL_SALES">=42)

Statistics

1 recursive calls
0 db block gets
134834 consistent gets
134809 physical reads
0 redo size
47506 bytes sent via SQL*Net to client
1987 bytes received via SQL*Net from client
135 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
2009 rows processed  => 1/50 rows
Storage Indexes – “Clustering Factor”

```
SQL> select name , value/1024/1024 MB from v$statname n, v$mystat s where n.statistic# = s.statistic# and n.name in ('cell physical IO interconnect bytes returned by smart scan', 'cell physical IO bytes saved by storage index');

Name                                                                                         MB
--------------------------------------------------------------------------------------
cell physical IO bytes saved by storage index                                               72.65625

cell physical IO interconnect bytes returned by smart scan                                 .383415222

• The Storage Index has only managed to avoid having to read approx. 7% of the table.
• The clustering of the data can make a huge impact to the effectiveness of Storage Indexes.
• Even though this query is only reading a fraction of the data as the previous query, it’s taking some 5 x longer.
• The smart scan though is only returning a small fraction of the overall data back to the database.
Exadata Storage Zone Maps

SQL> alter index big_bowie_album_id_i invisible;
Index altered.

SQL> create materialized zonemap big_bowie_album_id_zm on big_bowie(album_id);
Materialized zonemap created.
Exadata Storage Zone Maps

SQL> select * from big_bowie where album_id=42;
Elapsed: 00:00:00.28

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 STORAGE FULL WITH ZONEMAP</td>
<td>BIG_BOWIE</td>
<td>100K</td>
<td>8984K</td>
<td>3269 (12)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - storage("ALBUM_ID"=42)

filter(SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$", CASE WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE WHEN (zm."MIN_1_ALBUM_ID" > :1 OR zm."MAX_1_ALBUM_ID" < :2) THEN 3 ELSE 2 END END FROM "BOWIE"."BIG_BOWIE_ALBUM_ID_ZM" zm WHERE zm."ZONE_LEVEL$"=0 ORDER BY zm."ZONE_ID$",SYS_OP_ZONE_ID(ROWID),42,42)<3 AND "ALBUM_ID"=42)

Statistics

141 recursive calls
0 db block gets
2364 consistent gets
0 physical reads
0 redo size
3130019 bytes sent via SQL*Net to client
760 bytes received via SQL*Net from client
21 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
100000 rows processed
Exadata Storage Zone Maps

```sql
SQL> select segment_name, segment_type, blocks, bytes from dba_segments
   where segment_name like 'BIG_BOWIE%';

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>SEGMENT_TYPE</th>
<th>BLOCKS</th>
<th>BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG_BOWIE</td>
<td>TABLE</td>
<td>139264</td>
<td>1140850688</td>
</tr>
<tr>
<td>BIG_BOWIE_ALBUM_ID_I</td>
<td>INDEX</td>
<td>20480</td>
<td>167772160</td>
</tr>
<tr>
<td>BIG_BOWIE_ALBUM_ID_ZM</td>
<td>TABLE</td>
<td>8</td>
<td>65536</td>
</tr>
</tbody>
</table>

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

PL/SQL procedure successfully completed.

SQL> select blocks, num_rows from dba_tables where table_name='BIG_BOWIE_ALBUM_ID_ZM';

<table>
<thead>
<tr>
<th>BLOCKS</th>
<th>NUM_ROWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>134</td>
</tr>
</tbody>
</table>
```
### Exadata Storage Zone Maps

SQL> select zone_id$, min_1_album_id, max_1_album_id, zone_rows$ from big_bowie_album_id_zm;

<table>
<thead>
<tr>
<th>ZONE_ID$</th>
<th>MIN_1_ALBUM_ID</th>
<th>MAX_1_ALBUM_ID</th>
<th>ZONE_ROWS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>385855025152</td>
<td>1</td>
<td>2</td>
<td>66234</td>
</tr>
<tr>
<td>385855029250</td>
<td>5</td>
<td>6</td>
<td>56715</td>
</tr>
<tr>
<td>385855029251</td>
<td>7</td>
<td>7</td>
<td>76562</td>
</tr>
<tr>
<td>385855025155</td>
<td>7</td>
<td>8</td>
<td>76362</td>
</tr>
<tr>
<td>385855004675</td>
<td>8</td>
<td>9</td>
<td>76633</td>
</tr>
<tr>
<td>385855025161</td>
<td>21</td>
<td>22</td>
<td>75615</td>
</tr>
<tr>
<td>385855004684</td>
<td>29</td>
<td>29</td>
<td>75582</td>
</tr>
<tr>
<td>385855004685</td>
<td>31</td>
<td>32</td>
<td>75545</td>
</tr>
<tr>
<td>385855004687</td>
<td>35</td>
<td>36</td>
<td>75617</td>
</tr>
<tr>
<td>385855029267</td>
<td>42</td>
<td>43</td>
<td>75615</td>
</tr>
<tr>
<td>385855029270</td>
<td>50</td>
<td>50</td>
<td>75481</td>
</tr>
<tr>
<td>385855029275</td>
<td>61</td>
<td>62</td>
<td>75616</td>
</tr>
<tr>
<td>385855025179</td>
<td>62</td>
<td>63</td>
<td>75530</td>
</tr>
<tr>
<td>385855029284</td>
<td>81</td>
<td>82</td>
<td>75615</td>
</tr>
<tr>
<td>385855029285</td>
<td>84</td>
<td>84</td>
<td>75480</td>
</tr>
<tr>
<td>385855004710</td>
<td>87</td>
<td>88</td>
<td>75616</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385855029287</td>
<td>88</td>
<td>89</td>
<td>75618</td>
</tr>
<tr>
<td>385855004714</td>
<td>97</td>
<td>97</td>
<td>75771</td>
</tr>
<tr>
<td>385855029295</td>
<td>100</td>
<td>100</td>
<td>15871</td>
</tr>
</tbody>
</table>

134 rows selected.
Exadata Storage Zone Maps

SQL> create materialized zonemap big_bowie_artist_id_zm on big_bowie(artist_id);
create materialized zonemap big_bowie_artist_id_zm on big_bowie(artist_id)
ERROR at line 1:
ORA-31958: fact table "BOWIE"."BIG_BOWIE" already has a zonemap "BOWIE"."BIG_BOWIE_ALBUM_ID_ZM" on it

SQL> drop materialized zonemap big_bowie_album_id_zm;
Materialized zonemap dropped.

SQL> create materialized zonemap big_bowie_zm on big_bowie(album_id, artist_id);
Materialized zonemap created.

SQL> select measure, position_in_select, agg_function, agg_column_name
       from dba_zonemap_measures where zonemap_name='BIG_BOWIE_ZM';  2

<table>
<thead>
<tr>
<th>MEASURE</th>
<th>POSITION_IN_SELECT</th>
<th>AGG_FUNCTION</th>
<th>AGG_COLUMN_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BOWIE&quot;.&quot;BIG_BOWIE&quot;.&quot;ARTIST_ID&quot;</td>
<td>5</td>
<td>MAX</td>
<td>MAX_2_ARTIST_ID</td>
</tr>
<tr>
<td>&quot;BOWIE&quot;.&quot;BIG_BOWIE&quot;.&quot;ARTIST_ID&quot;</td>
<td>4</td>
<td>MIN</td>
<td>MIN_2_ARTIST_ID</td>
</tr>
<tr>
<td>&quot;BOWIE&quot;.&quot;BIG_BOWIE&quot;.&quot;ALBUM_ID&quot;</td>
<td>3</td>
<td>MAX</td>
<td>MAX_1_ALBUM_ID</td>
</tr>
<tr>
<td>&quot;BOWIE&quot;.&quot;BIG_BOWIE&quot;.&quot;ALBUM_ID&quot;</td>
<td>2</td>
<td>MIN</td>
<td>MIN_1_ALBUM_ID</td>
</tr>
</tbody>
</table>
# Exadata Storage Zone Maps

SQL> select zone_id$, min_1_album_id, max_1_album_id, zone_rows$ from big_bowie_zm;

<table>
<thead>
<tr>
<th>ZONE_ID$</th>
<th>MIN_1_ALBUM_ID</th>
<th>MAX_1_ALBUM_ID</th>
<th>ZONE_ROWS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>2</td>
<td>66234</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>5</td>
<td>6</td>
<td>56715</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>7</td>
<td>7</td>
<td>76562</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>7</td>
<td>8</td>
<td>76632</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>8</td>
<td>9</td>
<td>76633</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>21</td>
<td>22</td>
<td>75615</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>29</td>
<td>29</td>
<td>75582</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>31</td>
<td>32</td>
<td>75545</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>35</td>
<td>36</td>
<td>75617</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>43</td>
<td>44</td>
<td>75615</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>76</td>
<td>77</td>
<td>75615</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>79</td>
<td>80</td>
<td>75615</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>86</td>
<td>87</td>
<td>75616</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>88</td>
<td>89</td>
<td>75618</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>97</td>
<td>97</td>
<td>75771</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>100</td>
<td>100</td>
<td>15871</td>
</tr>
</tbody>
</table>

134 rows selected.
### Exadata Storage Zone Maps

```sql
SQL> select zone_id$, min_2_artist_id, max_2_artist_id, zone_rows$ from big_bowie_zm;
```

<table>
<thead>
<tr>
<th>ZONE_ID$</th>
<th>MIN_2_ARTIST_ID</th>
<th>MAX_2_ARTIST_ID</th>
<th>ZONE_ROWS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8586E+11</td>
<td>3661</td>
<td>98244</td>
<td>66234</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>56715</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>5273</td>
<td>81834</td>
<td>76562</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>76632</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>76632</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75615</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>2383</td>
<td>77964</td>
<td>75582</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75545</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75617</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75615</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75615</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75615</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75616</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75618</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>4848</td>
<td>80618</td>
<td>75771</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>84130</td>
<td>100000</td>
<td>15871</td>
</tr>
</tbody>
</table>
```

134 rows selected.
Exadata Storage Zone Maps

```sql
SQL> select * from big_bowie where artist_id=42;
Elapsed: 00:00:01.69

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>99</td>
<td>9108</td>
<td>3291 (13)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS STORAGE FULL WITH ZONEMAP</td>
<td>BIG_BOWIE</td>
<td>99</td>
<td>9108</td>
<td>3291 (13)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - storage("ARTIST_ID"=42)
   filter(SYS_ZMAP_FILTER("/* ZM_PRUNING */ SELECT "ZONE_ID$", CASE WHEN
       BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE WHEN (zm."MIN_2_ARTIST_ID" > :1 OR
       zm."MAX_2_ARTIST_ID" < :2) THEN 3 ELSE 2 END END FROM "BOWIE"."BIG_BOWIE_ZM" zm WHERE
       zm."ZONE_LEVEL$"=0 ORDER BY zm."ZONE_ID$",SYS_OP_ZONE_ID(ROWID),42,42)<3 AND "ARTIST_ID"=42)

Statistics

   141  recursive calls
   0    db block gets
   101614  consistent gets
   0    physical reads
   0    redo size
5190  bytes sent via SQL*Net to client
618   bytes received via SQL*Net from client
   8    SQL*Net roundtrips to/from client
   0    sorts (memory)
   0    sorts (disk)
100   rows processed
```
Exadata Storage Zone Maps

SQL> drop materialized zonemap big_bowie_zm;
Materialized zonemap dropped.

SQL> alter table big_bowie add clustering by linear order(artist_id, album_id) with materialized zonemap;
Table altered.

SQL> select zonemap_name from dba_zonemaps where fact_table='BIG_BOWIE';

<table>
<thead>
<tr>
<th>ZONEMAP_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZMAP$_BIG_BOWIE</td>
</tr>
</tbody>
</table>

SQL> select zonemap_name, pruning, with_clustering, invalid, stale, unusable from dba_zonemaps where zonemap_name = 'ZMAP$_BIG_BOWIE';

<table>
<thead>
<tr>
<th>ZONEMAP_NAME</th>
<th>PRUNING</th>
<th>WITH_CLUSTERING</th>
<th>INVALID</th>
<th>STALE</th>
<th>UNUSABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZMAP$_BIG_BOWIE</td>
<td>ENABLED</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>
### Exadata Storage Zone Maps

```sql
SQL> select zone_id$, min_1_artist_id, max_1_artist_id, zone_rows$ from zmap$_big_bowie;
```

<table>
<thead>
<tr>
<th>ZONE_ID$</th>
<th>MIN_1_ARTIST_ID</th>
<th>MAX_1_ARTIST_ID</th>
<th>ZONE_ROWS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75545</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75616</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75617</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75911</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75616</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75616</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75615</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75616</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75612</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75615</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75615</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75616</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>23432</td>
<td>98911</td>
<td>75480</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>1</td>
<td>100000</td>
<td>75791</td>
</tr>
<tr>
<td>3.8586E+11</td>
<td>21104</td>
<td>96583</td>
<td>75480</td>
</tr>
</tbody>
</table>

134 rows selected. => WHY NO CHANGE ???
Exadata Storage Zone Maps

SQL> alter table big_bowie move online;
Table altered.

SQL> select zone_id$, min_1_artist_id, max_1_artist_id, zone_rows$ from zmap$_.big_bowie;

<table>
<thead>
<tr>
<th>ZONE_ID$</th>
<th>MIN_1_ARTIST_ID</th>
<th>MAX_1_ARTIST_ID</th>
<th>ZONE_ROWS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9704E+11</td>
<td>67</td>
<td>1036</td>
<td>21185</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>2359</td>
<td>2453</td>
<td>9452</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>8341</td>
<td>9106</td>
<td>76516</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>18933</td>
<td>19688</td>
<td>75501</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>22708</td>
<td>23463</td>
<td>75497</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>26483</td>
<td>27238</td>
<td>75501</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>27238</td>
<td>27993</td>
<td>75499</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>33278</td>
<td>34033</td>
<td>75504</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>36674</td>
<td>40449</td>
<td>75500</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>38563</td>
<td>39318</td>
<td>75501</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>77824</td>
<td>78579</td>
<td>75501</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>82354</td>
<td>83109</td>
<td>75501</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>88394</td>
<td>89149</td>
<td>75501</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>93679</td>
<td>94434</td>
<td>75501</td>
</tr>
<tr>
<td>3.9704E+11</td>
<td>98211</td>
<td>98969</td>
<td>75794</td>
</tr>
</tbody>
</table>

144 rows selected.
Exadata Storage Zone Maps

```sql
SQL> select * from big_bowie where artist_id=42;
Elapsed: 00:00:00.02

Execution Plan
<p>|</p>
<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>99</td>
<td>9108</td>
<td>3291 (13)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 1</td>
<td>TABLE ACCESS STORAGE FULL WITH ZONEMAP</td>
<td>BIG_BOWIE</td>
<td>99</td>
<td>9108</td>
<td>3291 (13)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

1 - storage("ARTIST_ID"=42)
  filter(SYS_ZMAP_FILTER('/* ZM_PRUNING */ SELECT "ZONE_ID$", CASE WHEN BITAND(zm."ZONE_STATE$",1)=1 THEN 1 ELSE CASE WHEN (zm."MIN_1_ARTIST_ID" > :1 OR zm."MAX_1_ARTIST_ID" < :2) THEN 3 ELSE 2 END END FROM "BOWIE"."ZMAP$_BIG_BOWIE" zm WHERE zm."ZONE_LEVEL$"=0 ORDER BY zm."ZONE_ID$",SYS_OP_ZONE_ID(ROWID),42,42)<3 AND "ARTIST_ID"=42)

Statistics
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>recursive calls</td>
</tr>
<tr>
<td>db block gets</td>
</tr>
<tr>
<td>consistent gets</td>
</tr>
<tr>
<td>physical reads</td>
</tr>
<tr>
<td>redo size</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```
Conclusion: Before Creating A New Index...

- Consider Data Clustering
- Consider Database Constraints
- Consider Outlier Values
- Consider NULL Values
- Consider Partitioning
- Consider Partial Indexes
- Consider Efficiency Of Storage Indexes and Zone Maps
Introduction To Reverse Key Indexes: Part III (A Space Oddity) January 16, 2008

A possible 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 conflicts that take place.

Most Reverse key indices are created to resolve collision issues as a result of monotonically increasing values. As the database table values get inserted, each value is stored in the location defined in the index, and if there are no other values present, we fill the “right-most” block to the maximum inserted value. If an index performs 5-10 block optics meaning that full index scans are done, we will need to monitor and control buffer size.

However: it’s a way of removing duplicates will not hurt any more duplicated data in the table, but there will be a very remote chance of losing some of the maximum indexed value and 5-10 block optics meaning that full index scans are done, we will need to monitor and control buffer size. A Reverse Index is likely to be efficient when the “right-most” value is a point of ease.

If there are deletions, the story may differ.

Data 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 do not hit the leaf block, then the deleted space cannot 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 “street but not quite empty” index leaf blocks may in some scenarios increase to significant levels and this index may continue to grow at a greater proportional rate than the table (where the reuse of space is set and controlled by the PCTFREE algorithm amount).
Oracle Indexing Internals and Best Practices 5 Day Webinars

8-12 October, 6-10 November 2018 (4 Hours Daily)

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

Examines most available index structures/options & discusses in considerable detail how indexes function, how/when they should be used & how they should be maintained. Also how indexes are costed & evaluated by the Cost Based Optimizer (CBO) & how appropriate data management practices are vital for an effective indexing strategy. Covers many useful tips and strategies to maximise the benefits of indexes on application/database performance & scalability.

Running between 10am-2pm Zurich Time 8th – 12th October 2018
richardfooteconsulting.com/indexing-webinar/