```
*** First create a partitioned table, partitioned on a release_date column


SQL> CREATE TABLE big_album_sales(id number, album_id number, country_id number,

     release_date date, total_sales number) PARTITION BY RANGE (release_date)

    (PARTITION ALBUMS_2001 VALUES LESS THAN (TO_DATE('01-JAN-2002', 'DD-MON-YYYY')),

     PARTITION ALBUMS_2002 VALUES LESS THAN (TO_DATE('01-JAN-2003', 'DD-MON-YYYY')),

     PARTITION ALBUMS_2003 VALUES LESS THAN (TO_DATE('01-JAN-2004', 'DD-MON-YYYY')),

     PARTITION ALBUMS_2004 VALUES LESS THAN (TO_DATE('01-JAN-2005', 'DD-MON-YYYY')),

     PARTITION ALBUMS_2005 VALUES LESS THAN (TO_DATE('01-JAN-2006', 'DD-MON-YYYY')),

     PARTITION ALBUMS_2006 VALUES LESS THAN (TO_DATE('01-JAN-2007', 'DD-MON-YYYY')),

     PARTITION ALBUMS_2007 VALUES LESS THAN (TO_DATE('01-JAN-2008', 'DD-MON-YYYY')),

     PARTITION ALBUMS_2008 VALUES LESS THAN (MAXVALUE));

Table created.


*** Next populate the table with approximately 8 years worth of data

SQL> INSERT INTO big_album_sales SELECT rownum, mod(rownum,5000)+1, mod(rownum,100)+1, sysdate-
mod(rownum,2785), ceil(dbms_random.value(1,500000)) FROM dual CONNECT BY LEVEL <= 2000000;

2000000 rows created.

SQL> commit;

Commit complete.


*** Create an index on the release_date column

SQL> create index big_album_sales_i on big_album_sales(release_date) local;

Index created.

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

PL/SQL procedure successfully completed.


*** Create another equivalent table, this time non-partitioned

SQL> CREATE TABLE big_album_sales2(id number, album_id number, country_id number, release_date date,
total_sales number);

Table created.

SQL> INSERT INTO big_album_sales2 SELECT rownum, mod(rownum,5000)+1, mod(rownum,100)+1, sysdate-
mod(rownum,2785), ceil(dbms_random.value(1,500000)) FROM dual CONNECT BY LEVEL <= 2000000;

2000000 rows created.

SQL> commit;

Commit complete.

SQL> create index big_album_sales2_i on big_album_sales2(release_date);

Index created.

SQL> exec dbms_stats.gather_table_stats(ownname=> 'BOWIE', tabname=> 'BIG_ALBUM_SALES2',
estimate_percent=> null, method_opt=> 'FOR ALL COLUMNS SIZE 1');

PL/SQL procedure successfully completed.
```

```
SQL> set timing on

SQL> set arraysize 1000

SQL> set autotrace traceonly


*** Selecting 1 of the 8 years worth of data in the non-partitioned table will result in a full table scan
*** as it's the most efficient method in selecting such a relatively high percentage of rows

SQL> select * from big_album_sales2 where release_date between '01-JAN-2003' and '31-DEC-2003';

261352 rows selected.

Elapsed: 00:00:05.43

Execution Plan

----------------------------------------------------------

Plan hash value: 4100370891

-------------------------------------------------------------------------------------------

| Id  | Operation         | Name             | Rows  | Bytes | Cost (%CPU)| Time     |

-------------------------------------------------------------------------------------------

|   0 | SELECT STATEMENT  |                  |   262K|  6419K|  2304   (2)| 00:00:28 |

|*  1 |  TABLE ACCESS FULL| BIG_ALBUM_SALES2 |   262K|  6419K|  2304   (2)| 00:00:28 |

-------------------------------------------------------------------------------------------


Predicate Information (identified by operation id):

---------------------------------------------------

   1 - filter("RELEASE_DATE"<=TO_DATE(' 2003-12-31 00:00:00', 'syyyy-mm-dd

              hh24:mi:ss') AND "RELEASE_DATE">=TO_DATE(' 2003-01-01 00:00:00', 'syyyy-mm-dd

              hh24:mi:ss'))

Statistics

----------------------------------------------------------

          1  recursive calls

          0  db block gets

       8664  consistent gets

       8399  physical reads

          0  redo size

    6945261  bytes sent via SQL*Net to client

       3267  bytes received via SQL*Net from client

        263  SQL*Net roundtrips to/from client

          0  sorts (memory)

          0  sorts (disk)

     261352  rows processed


*** So that's 8664 consistent gets ...
```

*** Forcing the use of an index clearly shows is a more expensive option both in elapsed times and the increased consistent gets

```
SQL> select /*+ index (t) */ * from big_album_sales2 t where release_date between '01-JAN-2003' and '31-DEC-2003';

261352 rows selected.

Elapsed: 00:00:15.50

Execution Plan

----------------------------------------------------------

Plan hash value: 1720219014


-------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name               | Rows  | Bytes | Cost(%CPU) | Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                    |   262K|  6419K|   263K  (1)| 00:52:45 |
|   1 |  TABLE ACCESS BY INDEX ROWID| BIG_ALBUM_SALES2   |   262K|  6419K|   263K  (1)| 00:52:45 |
|*  2 |   INDEX RANGE SCAN          | BIG_ALBUM_SALES2_I |   262K|       |   701   (1)| 00:00:09 |
-------------------------------------------------------------------------------------------------------


Predicate Information (identified by operation id):

---------------------------------------------------


   2 - access("RELEASE_DATE">=TO_DATE(' 2003-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')

              AND "RELEASE_DATE"<=TO_DATE(' 2003-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))


Statistics

----------------------------------------------------------

          1  recursive calls

          0  db block gets

     262309  consistent gets

       2302  physical reads

          0  redo size

    5897978  bytes sent via SQL*Net to client

       3267  bytes received via SQL*Net from client

        263  SQL*Net roundtrips to/from client

          0  sorts (memory)

          0  sorts (disk)

     261352  rows processed
```

*** Consistent gets have increased significantly from 8664 to 262309 ...

*** Performing the same select on the partitioned table however is by far a cheaper option as the full table scan can take advantage of partition pruning, only having the visit partition 3

```
SQL> select * from big_album_sales where release_date between '01-JAN-2003' and '31-DEC-2003';

261352 rows selected.

Elapsed: 00:00:02.34


Execution Plan

----------------------------------------------------------

Plan hash value: 3245858454


-------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name            | Rows  | Bytes | Cost (%CPU)|Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                 |   262K|  6398K|   304   (2)|00:00:04 |       |       |
|   1 |  PARTITION RANGE SINGLE|                 |   262K|  6398K|   304   (2)|00:00:04 |     3 |     3 |
|*  2 |   TABLE ACCESS FULL    | BIG_ALBUM_SALES |   262K|  6398K|   304   (2)|00:00:04 |     3 |     3 |
-------------------------------------------------------------------------------------------------------------


Predicate Information (identified by operation id):

---------------------------------------------------


   2 - filter("RELEASE_DATE"<=TO_DATE(' 2003-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))


Statistics

----------------------------------------------------------

          1  recursive calls

          0  db block gets

       1366  consistent gets

       1008  physical reads

          0  redo size

    6945107  bytes sent via SQL*Net to client

       3267  bytes received via SQL*Net from client

        263  SQL*Net roundtrips to/from client

          0  sorts (memory)

          0  sorts (disk)

     261352  rows processed
```

*** Consistent gets have now dropped significantly from 8664 to just 1366 ...