```
*** Create table I usually use for Data Warehouse related demos

*** Oracle 10.2.0.3 with 8K block size in a non-ASSM tablespace


SQL> CREATE TABLE big_dwh_table (id NUMBER, album_id NUMBER, artist_id NUMBER, country_id NUMBER,
format_id number, release_date DATE, total_sales NUMBER);

Table created.


*** Create a little sequence and procedure to populate the table

SQL> CREATE SEQUENCE dwh_seq;

Sequence created.


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

Procedure created.

SQL> exec pop_big_dwh_table

PL/SQL procedure successfully completed.


*** Create a b-tree index on the ALBUM_ID column


SQL> CREATE INDEX big_dwh_album_id_i ON big_dwh_table(album_id);

Index created.


*** Let's collect some stats


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

PL/SQL procedure successfully completed.


*** Let's see just how big this index might be

SQL> SELECT index_name, distinct_keys, num_rows, leaf_blocks, blocks FROM user_indexes i,
user_segments s WHERE i.index_name = s.segment_name and i.index_name = 'BIG_DWH_ALBUM_ID_I';

INDEX_NAME                     DISTINCT_KEYS   NUM_ROWS LEAF_BLOCKS     BLOCKS
------------------------------ ------------- ---------- ----------- ----------
BIG_DWH_ALBUM_ID_I                     10000    1000000        2090       2176

1 row selected.


*** Note the column has 10,000 distinct keys and requires 2,090 leaf blocks. As expected, it also
has 1,000,000 index row entries



*** Let's write a simple, equality query that should use this index.


SQL> SELECT * FROM big_dwh_table WHERE album_id = 42;

100 rows selected.
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 278738356

-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Rows  | Bytes | Cost(%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |   100 |  3000 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| BIG_DWH_TABLE      |   100 |  3000 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | BIG_DWH_ALBUM_ID_I |   100 |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------


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

   2 - access("ALBUM_ID"=42)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         19  consistent gets
          0  physical reads
          0  redo size
       4649  bytes sent via SQL*Net to client
        462  bytes received via SQL*Net from client
          8  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        100  rows processed


*** OK, all looks fine. The index has been used as expected and we've only required 19 consistent
reads. Not bad at all.


*** Let's drop the b-tree index and create a bitmap index instead ...

SQL> drop index big_dwh_album_id_i;

Index dropped.


*** COMPUTE STATISTICS is redundant in 10g but there for backward compatibility ...

SQL> CREATE BITMAP INDEX big_dwh_album_id_i ON big_dwh_table(album_id) COMPUTE STATISTICS;

Index created.


*** Let's look at the index stats. With 10,000 distinct values, this index is likely to be huge,
right ?


SQL> SELECT index_name, distinct_keys, num_rows, leaf_blocks, blocks FROM user_indexes i,
user_segments s WHERE i.index_name = s.segment_name and i.index_name =  'BIG_DWH_ALBUM_ID_I';

INDEX_NAME                     DISTINCT_KEYS   NUM_ROWS LEAF_BLOCKS     BLOCKS
------------------------------ ------------- ---------- ----------- ----------
BIG_DWH_ALBUM_ID_I                     10000      10000          56        128

1 row selected.


*** Wait a minute. The number of leaf blocks has gone down (plummeted) from 2,090 to just 56.

*** Note also that the index only has 10,000 row entries and not the 1,000,000 as previous


*** How does the index impact our query ?


SQL> SELECT * FROM big_dwh_table WHERE album_id = 42;

100 rows selected.


Execution Plan
```

```
-------------------------------------------------------------
Plan hash value: 1178451901

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                    |   100 |  3000 |    22   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID | BIG_DWH_TABLE      |   100 |  3000 |    22   (0)| 00:00:01 |
|   2 |   BITMAP CONVERSION TO ROWIDS|                    |       |       |            |          |
|*  3 |    BITMAP INDEX SINGLE VALUE | BIG_DWH_ALBUM_ID_I |       |       |            |          |
-----------------------------------------------------------------------------------------------------------


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

   3 - access("ALBUM_ID"=42)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         11  consistent gets
          0  physical reads
          0  redo size
       4649  bytes sent via SQL*Net to client
        462  bytes received via SQL*Net from client
          8  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
        100  rows processed


*** CRs has dropped down to just 11 from 19 ...

*** Maybe a bitmap index isn't such a bad idea for columns with many distinct values after all ...
```