*** Create a simple little table called CASE_SEARCH with a column called NAME with lots of Unique values ...

```
SQL> CREATE TABLE case_search AS SELECT rownum id, 'DAVID BOWIE '|| rownum name FROM dual CONNECT BY level <= 100000;

Table created.
```

*** Now insert three new values for name with the same value EXCEPT they're not in the same "case".

```
SQL> INSERT INTO case_search VALUES (100001, 'Ziggy');

1 row created.

SQL> INSERT INTO case_search VALUES (100002, 'ZIGGY');

1 row created.

SQL> INSERT INTO case_search VALUES (100003, 'ZiGgY');

1 row created.

SQL> COMMIT;

Commit complete.
```

*** Now create a "normal" B-Tree index of the NAME column

```
SQL> CREATE INDEX case_search_name_i ON case_search(name);

Index created.
```

*** Collect statistics. No histograms required as the NAME column is effectively unique

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

PL/SQL procedure successfully completed.
```

*** Performing a search of the specific upper case version of "ZIGGY" will return only the one row as text searches by default are case sensitive

*** BUT will happily use the index due to the low cardinality of expected rows

```
SQL> SELECT * FROM case_search WHERE name = 'ZIGGY';

        ID NAME
---------- --------------------------------------------------
    100002 ZIGGY


Execution Plan
----------------------------------------------------------
Plan hash value: 2317465781

-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Rows  | Bytes | Cost(%CPU)| Time      |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |     1 |    22 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| CASE_SEARCH        |     1 |    22 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | CASE_SEARCH_NAME_I |     1 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------
```

*** So only one row selected but uses the index

*** We can try to select all the "Ziggy" values by converting all the values to UPPER case ...

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

        ID NAME
---------- --------------------------------------------------
    100001 Ziggy
    100002 ZIGGY
```

```
    100003 ZiGgY


Execution Plan
----------------------------------------------------------
Plan hash value: 3830515511

--------------------------------------------------------------------------------
| Id  | Operation         | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |             |  1000 | 22000 |    90   (4)| 00:00:02|
|*  1 |  TABLE ACCESS FULL| CASE_SEARCH |  1000 | 22000 |    90   (4)| 00:00:02|
--------------------------------------------------------------------------------



*** We now select all 3 rows but it doesn't use the index as the UPPER function negates the use of
the index

*** A common strategy is to therefore create a Function-Based Index on the UPPER(NAME) column ...


SQL> CREATE INDEX case_search_upper_name_i ON case_search(UPPER(name)) COMPUTE STATISTICS;

Index created.

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

        ID NAME
---------- --------------------------------------------------
    100001 Ziggy
    100002 ZIGGY
    100003 ZiGgY


Execution Plan
----------------------------------------------------------
Plan hash value: 3980014320

--------------------------------------------------------------------------------------------------------
----
| Id  | Operation                   | Name                      | Rows  | Bytes | Cost (%CPU)| Time
|
--------------------------------------------------------------------------------------------------------
----
|   0 | SELECT STATEMENT            |                           |  1000 | 22000 |    81   (0)|
00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| CASE_SEARCH               |  1000 | 22000 |    81   (0)|
00:00:01 |
|*  2 |   INDEX RANGE SCAN          | CASE_SEARCH_UPPER_NAME_I |   400 |       |     3   (0)|
00:00:01 |
--------------------------------------------------------------------------------------------------------
----



*** It now returns all 3 rows and uses an index but the application needs to be written with the
appropriate function call


*** Let's now create a Linguistic Index.

*** We sort the index entries using the NLS_GENERIC_M_CI option (GENERIC_M is the standard generic
sorting order for Latin based characters, _CI for case-Insensitive sorts/searches)

*** Note: a language where the basic BINARY order is sufficient, the NLS_SORT=BINARY_CI option can
be used for case insensitive searches.

SQL> CREATE INDEX case_search_ling_name_i ON case_search(NLSSORT(name,'NLS_SORT=GENERIC_M_CI'));

Index created.


*** Alter the session to use the required linguistic sorting order

SQL> ALTER SESSION SET NLS_SORT='GENERIC_M_CI';

Session altered.


*** Alter the session to enable linguistic sorts/searches
```

```
SQL> ALTER SESSION SET NLS_COMP='LINGUISTIC';

Session altered.


*** Now let's try and retrieve all the Ziggy versions without having to change the application code

SQL> SELECT * FROM case_search WHERE name = 'ZIGGY';

        ID NAME
---------- --------------------------------------------------
    100001 Ziggy
    100002 ZIGGY
    100003 ZiGgY


Execution Plan
----------------------------------------------------------
Plan hash value: 1488392692

-------------------------------------------------------------------------------------------------------
---
| Id  | Operation                   | Name                     | Rows  | Bytes |Cost (%CPU)| Time
|
-------------------------------------------------------------------------------------------------------
---
|   0 | SELECT STATEMENT            |                          |  1000 | 23000 |   81   (0)| 00:00:01
|
|   1 |  TABLE ACCESS BY INDEX ROWID| CASE_SEARCH              |  1000 | 23000 |   81   (0)| 00:00:01
|
|*  2 |   INDEX RANGE SCAN          | CASE_SEARCH_LING_NAME_I |   400 |       |    3   (0)| 00:00:01
|
-------------------------------------------------------------------------------------------------------
---


*** Note we now get all 3 rows, via an index without the need for the application to call a specific
function ...


*** BUT there are a number of significant issues to be covered later...
```