*** First, create table. Note that the ID column is really really selective but the CODE column has just the 5 distinct values

```
SQL> CREATE TABLE ziggy_stuff AS SELECT mod(rownum,500000) id, mod(rownum,5) code, 'ZIGGY'
name FROM dual CONNECT BY LEVEL <= 1000000;

Table created.
```

*** First create an index with the leading column being the low cardinality, just 5 distinct values CODE column, followed by the really selective ID column

*** This index combination is going to be really inefficient to use right ...

```
SQL> CREATE INDEX ziggy_stuff_i ON ziggy_stuff(code, id);

Index created.

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

PL/SQL procedure successfully completed.
```

*** Let's look at just how good this index really is.

```
SQL> SELECT * FROM ziggy_stuff WHERE id = 4242 and code = 2;


Execution Plan
----------------------------------------------------------
Plan hash value: 2876492483

---------------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |     1 |    13 |     5   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ZIGGY_STUFF   |     1 |    13 |     5   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | ZIGGY_STUFF_I |     2 |       |     3   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------


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

   2 - access("CODE"=2 AND "ID"=4242)


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

*** Actually that's not bad at all. A cost of just 5 and 6 CRs seems perfectly reasonable

*** The BLEVEL of the index is 2 so that's 4 CRs to get to down to the leaf block and read the 2 index entries, 2 CRs for the visit to the table ...

*** How does Oracle get to the exact leaf block of interest, why doesn't it have to plough through a whole bunch of CODEs with a value of 2 to get to the ID of interest ?

*** Well a block dump of a root branch block reveals the answer ...

```
Block header dump:  0x0480828a
 Object id on Block? Y
 seg/obj: 0x1418f  csc: 0x01.e2b2e033  itc: 1  flg: -  typ: 2 - INDEX
     fsl: 0  fnx: 0x0 ver: 0x01

 Itl           Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0xffff.000.00000000  0x00000000.0000.00  C---    0  scn 0x0001.e2b2e033

Branch block dump
=================
header address 4137540=0x3f2244
kdxcolev 2
KDXCOLEV Flags = - - -
kdxcolok 0
```

```
kdxcoopc 0x80: opcode=0: iot flags=--- is converted=Y
kdxconco 3
kdxcosdc 0
kdxconro 4
kdxcofbo 36=0x24
kdxcofeo 8008=0x1f48
kdxcoavs 7972
kdxbrlmc 75531717=0x48085c5            ==> pointer (data block address) to the first
intermediate branch block
kdxbrsno 0
kdxbrbksz 8060
kdxbr2urrc 0
row#0[8047] dba: 75532512=0x48088e0    ==> pointer to the second intermediate branch block
col 0; len 2; (2):  c1 02              ==> first column (CODE) value by which to navigate
col 1; len 4; (4):  c3 07 3f 39        ==> second column (ID) value by which to navigate.
Index entries less than these values go via the first branch block
col 2; TERM                            ==> third column (ROWID) is not required to identify
the necessary path and so the branch entry is terminated at this point
row#1[8034] dba: 75537018=0x4809a7a    ==> pointer to the third intermediate branch block
(and so on ....)
col 0; len 2; (2):  c1 03
col 1; len 4; (4):  c3 08 34 62
col 2; TERM
row#2[8021] dba: 75537812=0x4809d94
col 0; len 2; (2):  c1 04
col 1; len 4; (4):  c3 09 20 5e
col 2; TERM
row#3[8008] dba: 75563461=0x48101c5
col 0; len 2; (2):  c1 05
col 1; len 4; (4):  c3 0a 0c 5a
col 2; TERM
----- end of branch block dump -----
End dump data blocks tsn: 21 file#: 18 minblk 33418 maxblk 33418



*** So the branch nodes contain columns values of all the indexed columns required to
identify a unique path by which any index entry can be direct to the relevant leaf block

*** That being the case, the order of the index columns on it's own should make little
difference to the performance of the index


*** Let's see if there indeed is any difference ...

SQL> drop index ziggy_stuff_i;

Index dropped.


*** Now create the index with the columns the other way around

SQL> CREATE INDEX ziggy_stuff_i ON ziggy_stuff(id, code);

Index created.


*** Same query ...

SQL> SELECT * FROM ziggy_stuff WHERE id = 4242 and code = 2;


Execution Plan
----------------------------------------------------------
Plan hash value: 2876492483

-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |     1 |    13 |     5   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| ZIGGY_STUFF   |     1 |    13 |     5   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | ZIGGY_STUFF_I |     2 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------


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

   2 - access("ID"=4242 AND "CODE"=2)


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


*** Identical index performance ...



*** Block dump of branch (root) block ...

Block header dump:  0x0480828a
 Object id on Block? Y
 seg/obj: 0x14190  csc: 0x01.e2b2e1c7  itc: 1  flg: -  typ: 2 - INDEX
     fsl: 0  fnx: 0x0 ver: 0x01

 Itl           Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0xffff.000.00000000  0x00000000.0000.00  C---    0  scn 0x0001.e2b2e1c7

Branch block dump
=================
header address 4137540=0x3f2244
kdxcolev 2
KDXCOLEV Flags = - - -
kdxcolok 0
kdxcoopc 0x80: opcode=0: iot flags=--- is converted=Y
kdxconco 3
kdxcosdc 0
kdxconro 3
kdxcofbo 34=0x22
kdxcofeo 8030=0x1f5e
kdxcoavs 7996
kdxbrlmc 75531938=0x48086a2
kdxbrsno 0
kdxbrbksz 8060
kdxbr2urrc 0
row#0[8050] dba: 75532865=0x4808a41      ==> pointer to the second intermediate branch block
col 0; len 4; (4):  c3 0d 46 46          ==> first column (ID) value by which to navigate,
note that as it has such a high cardinality ...
col 1; TERM                              ==> the second column (CODE) is not required to
identify the necessary path and so the branch entry is terminated at this point
row#1[8040] dba: 75537504=0x4809c60
col 0; len 4; (4):  c3 1a 32 28
col 1; TERM
row#2[8030] dba: 75563462=0x48101c6
col 0; len 4; (4):  c3 27 1e 0a
col 1; TERM
----- end of branch block dump -----
End dump data blocks tsn: 21 file#: 18 minblk 33418 maxblk 33418


*** So the index performed in exactly the same manner for both indexes ...
```