Example 11:
cqlsh
Restrictions on clustering keys

Let's see the skus CF description

```
cassandra@cqlsh:catalog> describe skus;

CREATE TABLE catalog.skus (
    sellerid text,
    skuid text,
    timeofskucreation text,
    productid text,
    islistingcreated boolean,
    listingid text,
    title text,
    PRIMARY KEY ((sellerid, skuid), timeofskucreation, productid)
) WITH CLUSTERING ORDER BY (timeofskucreation ASC, productid ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class':
'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

Let's see the skus CF description

```
cassandra@cqlsh:catalog> describe skus;

CREATE TABLE catalog.skus (
    sellerid text,
    skuid text,
    timeofskucreation text,
    productid text,
    islistingcreated boolean,
    listingid text,
    title text,
    PRIMARY KEY ((sellerid, skuid), timeofskucreation, productid)
) WITH CLUSTERING ORDER BY (timeofskucreation ASC, productid ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class':
'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

data within a partition key will be ordered first by timeofskucreation and then by productId in ASC order

Let's query skus for Fab seller and sku FABSKU

```
cassandra@cqlsh:catalog> select * from skus where sellerid = 'Fab' and
skuid = 'FABSKU';

 sellerid | skuid  | timeofskucreation | productid | islistingcreated | listingid
      | title
----------+--------+-------------------+-----------+------------------+-----------
-----+----------------------
      Fab | FABSKU |  2015-12-11 12:21 |     SOFA1 |             True | LISTINGFabS
OFA1 |    Urban Living Derby
      Fab | FABSKU |  2015-12-11 12:21 |     SOFA3 |            False |
null | Urban Living 4 seater
```

timeofskucreation is same for both the rows
the row with SOFA1 comes before row with SOFA3

Let's query skus for Fab seller and sku FABSKU

```
cassandra@cqlsh:catalog> select * from skus where sellerid = 'Fab' and
skuid = 'FABSKU';

 sellerid | skuid  | timeofskucreation | productid | islistingcreated | listingid
     | title
----------+--------+-------------------+-----------+------------------+-----------
-----+----------------------
      Fab | FABSKU |  2015-12-11 12:21 |     SOFA1 |             True | LISTINGFabS
OFA1 |    Urban Living Derby
      Fab | FABSKU |  2015-12-11 12:21 |     SOFA3 |            False |
null | Urban Living 4 seater
```

A more accurate way to represent how data is stored is a nested structure

```
{
 'timeofskucreation':'2015-12-11 12:21'
        {'productid':'SOFA1'
                        {
                        'islistingcreated':'True',
                        'listingid':'LISTINGFabSOFA1'
                        'title':'Urban Living Derby'
                        }
          'productid':'SOFA3'
                         {
                         'islistingcreated':'False',
                         'title':'Urban Living 4 Seater'
                         }
         }

'timeofskucreation':'2016-01-01 00:01'
                    . . .
```

1st clustering column is the outermost column of the nested structure


```
{
 'timeofskucreation':'2016-01-01 00:01'
        {'productid':'SOFA1'
                'islistingcreated':'True',
                'title':'Urban Living Derby'
                }
        'productid':'SOFA3'
                {
                'islistingcreated':'False',
                'title':'Urban Living 4 Seater'
                }
        }
'timeofskucreation':'2016-01-01 00:01'
                . . .
```

```
{
 'timeofskucreation':'2015-12-11 12:21'
        {'productid':'SOFA1'
                        {
                        'islistingcreated':'True',
                        'listingid':'LISTINGFabSOFA1'
                        'title':'Urban Living 3 Seater'
                        }
          'productid':'SOFA3'
                        {
                        'islistingcreated':'False',
                        'title':'Urban Living 4 Seater'
                        }
         }

'timeofskucreation':'2016-01-01 00:01'
                    . . .
```

timeofskucreation is the first clustering column

```
{
 'timeofskucreation':'2015-12-11 12:21'
        {'productid':'SOFA1'
                        {
                        'islistingcreated':'True',
                        'listingid':'LISTINGFabSOFA1'
                        'title':'Urban Swing Derby'
                        }
         'productid':'SOFA3'
                        {
                        'islistingcreated':'False',
                        'title':'Urban Swing Derby'
                        }
        }

'timeofskucreation':'2016-01-01 00:01'
                    . . .
```

Next layer is the 2nd clustering column ie productid

```
{
 'timeofskucreation':'2015-12-11 12:21'
        {'productid':'SOFA1'
                        {
                        'islistingcreated':'True',
                        'listingid':'LISTINGFabSOFA1'
                        'title':'Urban Living Derby'
                        }
         'productid':'SOFA3'
                         {
                         'islistingcreated':'False',
                         'title':'Urban Living 4 Seater'
                         }
        }
'timeofskucreation':'2016-01-01 00:01'
                . . .
```

All the remaining columns are stored in the layer below productid

In order to retrieve data efficiently, we need to restrict the clustering columns in the order in which they are defined

To prevent inefficient queries, cassandra doesn't allow out of order restrictions on clustering columns

Let's understand this with an example

Let's put a restriction on the productid column

```
WITH CLUSTERING ORDER BY (timeofskucreation ASC,
productid ASC)
```

```
cassandra@cqlsh:catalog> select * from skus where sellerid = 'Fab' and skuid i
('FABSKU', 'FABSKU1') and productid = 'SOFA1';
```

```
InvalidRequest: code=2200 [Invalid query] message="PRIMARY KEY column "productid"
cannot be restricted as preceding column "timeofskucreation" is not restricted"
```

The query fails as we haven't restricted timeofskucreation column

Clustering column restrictions

If you want to restrict the nth clustering column, you need to restrict all the preceding 1 to n-1 clustering columns

For the query to work we need to restrict timeofskucreation

```
cassandra@cqlsh:catalog> select * from skus where sellerid =
'Fab' and skuid in ('FABSKU', 'FABSKU1') and
timeofskucreation = '2015-12-11 12:21' and productid =
'SOFA1';
```

```
 sellerid | skuid  | timeofskucreation | productid | islistingcreated | listingid
     | title
----------+--------+-------------------+-----------+------------------+-----------
-----+--------------------
      Fab | FABSKU |  2015-12-11 12:21 |     SOFA1 |             True | LISTINGFabS
OFA1 | Urban Living Derby
```

Restricting clustering columns with IN

```
cassandra@cqlsh:catalog> select * from skus where sellerid in ('Decor', 'Fab') and
skuid IN('FABSKU', 'DECORSKU') AND timeofskucreation IN ('2015-12-11 12:21', '2016-
07-01 22:30') and productid IN ('SOFA1', 'SOFA2');

 sellerid | skuid    | timeofskucreation | productid | islistingcreated | listingid
        | title
----------+----------+-------------------+-----------+------------------+---------
-------+--------------------
    Decor | DECORSKU |  2016-07-01 22:30 |     SOFA1 |            False |
  null | Urban Living Derby
      Fab |   FABSKU |  2015-12-11 12:21 |     SOFA1 |             True | LISTINGFa
bSOFA1 | Urban Living Derby

(2 rows)
```

Similarly IN query works if both the clustering columns are restricted

Clustering column restrictions

If you want to restrict the nth clustering column, you need to restrict all the preceding 1 to n-1 clustering columns

IN and = operators can be used with all the clustering columns

Let's restrict the clustering columns with range operators

```
cassandra@cqlsh:catalog> select * from skus where sellerid in ('Decor', 'Fab') and
skuid IN('FABSKU', 'DECORSKU') AND timeofskucreation >= '2015-08-01' and timeofskuc
reation <= '2016-07-01 22:30';

 sellerid | skuid     | timeofskucreation | productid | islistingcreated | listingid
        | title
----------+-----------+-------------------+-----------+------------------+----------
-------+--------------------
    Decor | DECORSKU |  2016-07-01 22:30 |     SOFA1 |            False |
  null | Urban Living Derby
      Fab |    FABSKU |  2015-12-11 12:21 |     SOFA1 |             True | LISTINGFa
bSOFA1 | Urban Living Derby

(2 rows)
```

Here productid is not restricted, which works perfectly fine as data is sorted by timeofskucreation

Let's restrict the clustering columns with range operators

```
cassandra@cqlsh:catalog> select * from skus where sellerid in ('Decor', 'Fab') and
skuid IN('FABSKU', 'DECORSKU') AND timeofskucreation >= '2015-08-01' and timeofskuc
reation <= '2016-07-01 22:30' and productid IN ('SOFA1');
```

```
InvalidRequest: code=2200 [Invalid query] message="Clustering column "productid"
cannot be restricted (preceding column "timeofskucreation" is restricted by a non-EQ
relation)"
```

This query could result in scanning of entire data on a single node

But it will lead to read timeouts if the partitions are on different nodes

Clustering column restrictions

If you want to restrict nth clustering column, you need to restrict all the preceding 1 to n-1 clustering columns

IN and = operators can be used with all the clustering columns

Clustering column cannot be restricted if its preceding column is restricted by a range operator