Example 12:
cqlsh
index

Let's create a columnfamily for seller SKU data

```
cassandra@cqlsh:catalog>CREATE COLUMNFAMILY listings (
  listingId varchar,
  sellerId varchar,
  skuId varchar,
  productId varchar,
  mrp int,
  ssp int,
  sla int,
  stock int,
  title text,
  PRIMARY KEY  ((sellerId,skuId), stock, ssp));
```

A list of products that a seller has on an e-commerce site

Let's create a columnfamily for seller SKU data

```
cassandra@cqlsh:catalog>CREATE COLUMNFAMILY listings (
  listingId varchar,
  sellerId varchar,
  skuId varchar,
  productId varchar,
  mrp int,
  ssp int,
  sla int,
  stock int,
  title text,
  PRIMARY KEY  ((sellerId,skuId), stock, ssp));
```

sellerid, sku is the partition key

Let's create a columnfamily for seller SKU data

```
cassandra@cqlsh:catalog>CREATE COLUMNFAMILY listings (
  listingId varchar,
  sellerId varchar,
  skuId varchar,
  productId varchar,
  mrp int,
  ssp int,
  sla int,
  stock int,
  title text,
  PRIMARY KEY  ((sellerId,skuId), stock, ssp));
```

stock and ssp are the clustering columns

We have some data for listings in csv file

```
Fab,12,5000,LISTINGSOFA1,6000,SOFA1,SKU1,5,Urban Living Derby
Decor,0,12000,LISTINGSOFA2,15000,SOFA2,SKU2,3,Urban Decor 2 seater
Chroma,100,20000,LISTINGCOM1,25000,COM1,SKUCOM1,2,Acer One
Chroma,150,21000,LISTINGCOM2,23000,COM2,SKUCOM2,2,Acer One
Chroma,100,20500,LISTINGCOM3,23000,COM3,SKUCOM1,3,Acer Plus
Fab,1,13000,LISTINGFABSOFA2,15000,SOFA2,SKU11,3,Urban 2 seater
```

We will populate data from csv file using COPY

```
cassandra@cqlsh:cms> COPY listings(sellerid,stock,ssp,listingid,mrp,productid,skuid
,sla,title) FROM 'listing.csv' ;
```

```
cassandra@cqlsh:catalog> select * from listings;

 sellerid | skuid    | stock | ssp   | listingid        | mrp   | productid | sla | t
itle
----------+----------+-------+-------+------------------+-------+-----------+-----+--
--------------------
    Chroma | SKUCOM1 |   100 | 20000 |      LISTINGCOM1 | 25000 |       COM1 |   2 |
            Acer One
    Chroma | SKUCOM1 |   100 | 20500 |      LISTINGCOM3 | 23000 |       COM3 |   3 |
           Acer Plus
       Fab |    SKU1 |    12 |  5000 |     LISTINGSOFA1 |  6000 |      SOFA1 |   5 |
 Urban Living Derby
    Chroma | SKUCOM2 |   150 | 21000 |      LISTINGCOM2 | 23000 |       COM2 |   2 |
            Acer One
     Decor |    SKU2 |     0 | 12000 |     LISTINGSOFA2 | 15000 |      SOFA2 |   3 | U
rban Decor 2 seater
       Fab |   SKU11 |     1 | 13000 | LISTINGFABSOFA2 | 15000 |      SOFA2 |   3 |
      Urban 2 seater

(6 rows)
```

data is imported in the listings CF

After all the partition and clustering key restrictions

Some of the operations that we can do for a seller

- List all the products by seller
- List products that have no stock
- List products that have a seller price in a certain range subject to stock conditions

All of these queries use primary keys

But what if..

we want to list the sellers who sell a group of products?

Or display any other data which is based on product ids rather than seller information?

For such use cases which
are beyond primary keys

We use the
Secondary Index

Secondary Index

are used to access data using
a non-primary column

secondary indexes are stored in
a separate columnfamily (CF)

best used when the indexed column
does not have a very high cardinality

Secondary Index cardinality

This refers to how many distinct values a column holds

A column should not have too few distinct values e.g. a column of boolean values

A column should not have too many distinct values e.g. a column where every value is unique

Secondary Index cardinality

too few distinct values

too many distinct values

low cardinality

high cardinality

Secondary Index

are used to access data using
a non-primary column

secondary indexes are stored in
a separate columnfamily (CF)

best used when the indexed column
does not have a very high cardinality

Secondary Index

Indexes should be used cautiously

They are not replicated to other nodes

Every query on the secondary index, the read request is forwarded to all the nodes

results are then merged and returned to client

As the size of the cluster increases, index queries become slower

Secondary Index

Indexes should never be used

on columns with high cardinality
i.e more distinct values

since we then query the cluster
for very few rows

on CFs that are updated frequently

on counter columns

Secondary Index

Lets create an index on productid

```
cassandra@cqlsh:catalog> create index listingprodindex on
listings(productid);
```

Secondary Index

```
cassandra@cqlsh:catalog> create index listingprodindex on
listings(productid);
```

name of the index

Secondary Index

```
cassandra@cqlsh:catalog> create index listingprodindex on
listings(productid);
```

on the productid column in the listings columnfamily

Secondary Index

We have seen that there are a few limitations on partition key usage

RECAP

Partition key Restrictions

All the columns of the partition key should be restricted in the query

We cannot use > , >=, <=, < operator directly on the partition key

Only IN and = operators are allowed on the partition key

We can use > , >=, <=, < operator on the token

ORDER BY is not supported with partition key

Secondary Index

Let's see the behaviour of index with unrestricted partition keys

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2';
```

Secondary Index

Let's see the behaviour of index with unrestricted partition keys

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2';
```

```
 sellerid | skuid | stock | ssp   | listingid        | mrp   | productid | sla | tit
le
----------+-------+-------+-------+------------------+-------+-----------+-----+----
-----------------
    Decor |  SKU2 |     0 | 12000 |    LISTINGSOFA2 | 15000 |     SOFA2 |   3 | Urb
an Decor 2 seater
      Fab | SKU11 |     1 | 13000 | LISTINGFABSOFA2 | 15000 |     SOFA2 |   3 |
   Urban 2 seater

(2 rows)
```

Secondary Index

With unrestricted partition keys

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2';

 sellerid | skuid | stock | ssp   | listingid        | mrp   | productid | sla | tit
le
----------+-------+-------+-------+------------------+-------+-----------+-----+----
-----------------
    Decor |  SKU2 |     0 | 12000 |     LISTINGSOFA2 | 15000 |     SOFA2 |   3 | Urb
an Decor 2 seater
      Fab | SKU11 |     1 | 13000 | LISTINGFABSOFA2 | 15000 |     SOFA2 |   3 |
   Urban 2 seater

(2 rows)
```

IT WORKS!

Secondary Index

With unrestricted partition keys

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2';

 sellerid | skuid | stock | ssp   | listingid        | mrp   | productid | sla | tit
le
----------+-------+-------+-------+------------------+-------+-----------+-----+----
----------------
    Decor |  SKU2 |     0 | 12000 | LISTINGDECORSOFA2 | 15000 |     SOFA2 |   3 | Urb
an Decor 2 seater
      Fab | SKU11 |     1 | 13000 | LISTINGFABSOFA2 | 15000 |     SOFA2 |   3 |
   Urban 2 seater

(2 rows)
```

No partition key in the where clause!

Partition key Restrictions

All the columns of the partition key should be restricted in the query unless we use a secondary index

We cannot use > , >=, <=, < operator directly on the partition key

Only IN and = operators are allowed on the partition key

We can use > , >=, <=, < operator on the token

ORDER BY is not supported with partition key

Secondary Index

Let's query with productid and sellerid

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2' and sellerid = 'Fab';
```

```
InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it
might involve data filtering and thus may have unpredictable performance. If you wa
nt to execute this query despite the performance unpredictability, use ALLOW FILTER
ING"
```

Secondary Index

Let's query with productid and sellerid

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2' and sellerid = 'Fab';

InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

Notice only sellerid is present in the restriction

Secondary Index

Let's query with productid and sellerid

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2' and sellerid = 'Fab';
```

```
InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it
might involve data filtering and thus may have unpredictable performance. If you wa
nt to execute this query despite the performance unpredictability, use ALLOW FILTER
ING"
```

So we still cannot query with only a part of partition key restricted

Secondary Index

Let's query with index and restricted partition key

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2' and sellerid = ‘Decor' and skuid =
'SKU2';
```

```
 sellerid | skuid | stock | ssp   | listingid     | mrp   | productid | sla | title
----------+-------+-------+-------+---------------+-------+-----------+-----+-------
--------------
    Decor |  SKU2 |     0 | 12000 | LISTINGSOFA2 | 15000 |     SOFA2 |   3 | Urban
Decor 2 seater

(1 rows)
```

Secondary Index

Let's query with index and restricted partition key

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2' and sellerid = ‘Decor' and skuid =
'SKU2';
```

```
 sellerid | skuid | stock | ssp   | listingid     | mrp   | productid | sla | title
----------+-------+-------+-------+---------------+-------+-----------+-----+-------
--------------
    Decor |  SKU2 |     0 | 12000 | LISTINGSOFA2 | 15000 |     SOFA2 |   3 | Urban
Decor 2 seater

(1 rows)
```

Works!

Secondary Index

Let's query with index and restricted partition key

```
cassandra@cqlsh:catalog> select * from listings where
productid='SOFA2' and sellerid='Decor' and sku =
'SKU2';

 sellerid | sku | price | qty | listingid | name | productid | sale | title
----------+-----+-------+-----+-----------+------+-----------+------+-------
-------------

    Decor |  SKU2 |     0 | 12000 | LISTINGSOFA2 | 15000 |     SOFA2 |   3 | Urban
Decor 2 seater

(1 rows)
```

With a secondary index, either restrict all partition keys or do not restrict partition keys at all

Index Key Restrictions

With index, either all the columns of the partition key should be restricted in the query or none

Secondary Index

Let's restrict clustering columns with the IN operator

```
cassandra@cqlsh:catalog> select * from listings where
productid IN (‘SOFA2’,'SOFA1') and ssp > 1000;
```

```
InvalidRequest: code=2200 [Invalid query] message="IN
predicates on non-primary-key columns (productid) is not
yet supported"
```

Secondary Index

Let's restrict clustering columns with the IN operator

```
cassandra@cqlsh:catalog> select * from listings where
productid IN (‘SOFA2’,'SOFA1') and ssp > 1000;

InvalidRequest: code=2200 [Invalid query] message="IN
predicates on non-primary-key columns (productid) is not
yet supported"
```

The IN operator is not supported yet for secondary indexes

Index Key Restrictions

With index, either all the columns of the partition key should be restricted in the query or none

We cannot use the IN predicate on indexed column

Secondary Index

Let's restrict indexed columns with a range operator

```
cassandra@cqlsh:catalog> select * from listings where
productid > 'SOFA';
```

```
InvalidRequest: code=2200 [Invalid query] message="Cannot
execute this query as it might involve data filtering and
thus may have unpredictable performance. If you want to
execute this query despite the performance
unpredictability, use ALLOW FILTERING"
```

Secondary Index

Let's restrict indexed columns with a range operator

```
cassandra@cqlsh:catalog> select * from listings where
productid > 'SOFA';

InvalidRequest: code=2200 [Invalid query] message="Cannot
execute this query as it might involve data filtering and
thus may have unpredictable performance. If you want to
execute this query despite the performance
unpredictability, use ALLOW FILTERING"
```

ALLOW FILTERING will enable the query to scan entire index

Secondary Index

Lets restrict indexed columns with range operator

```
cassandra@cqlsh:catalog> select * from listings where
productid > 'SOFA';
```

InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"

query is a valid query, but its performance will be poor

Index Key Restrictions

With index, either all the columns of the partition key should be restricted in the query or none

We cannot use the IN predicate on indexed column

Range operators >,<>=,<= are not preferred

Secondary Index

Let's restrict clustering columns with EQ

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2' and ssp = 12000 and stock = 0;
```

InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"

Secondary Index

Let's restrict clustering columns with EQ

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2' and ssp = 12000 and stock = 0;

InvalidRequest: code=2200 [Invalid query] message="Cannot
execute this query as it might involve data filtering and
thus may have unpredictable performance. If you want to
execute this query despite the performance
unpredictability, use ALLOW FILTERING"
```

Secondary Index

Let's restrict clustering columns with EQ

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA2' and ssp = 12000 and stock = 0;

InvalidRequest: code=2200 [Invalid query] message="Cannot
execute this query as it might involve data filtering and
thus may have unpredictable performance. If you want to
execute this query despite the performance
unpredictability, use ALLOW FILTERING"
```

Using restricted clustering column with index is not preferred

Index Key Restrictions

With index, either all the columns of the partition key should be restricted in the query or none

We cannot use the IN predicate on indexed column

Range operators >,<>=,<= are not preferred

Use of clustering columns with index is not preferred

Secondary Index

Query with ORDER BY

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA1' order by stock;
```

```
InvalidRequest: code=2200 [Invalid query] message="ORDER BY
with 2ndary indexes is not supported."
```

Secondary Index

Query with ORDER BY

```
cassandra@cqlsh:catalog> select * from listings where
productid = 'SOFA1' order by stock;
```

```
InvalidRequest: code=2200 [Invalid query] message="ORDER BY
with 2ndary indexes is not supported."
```

Order BY is not supported

Index Key Restrictions

With index, either all the columns of the partition key should be restricted in the query or none

We cannot use the IN predicate on indexed column

Range operators >,<>=,<= are not supported

Use of clustering columns with index is not allowed

ORDER BY is not supported

Secondary Index

Query with Collections

Let's add a column of type set

```
cassandra@cqlsh:catalog>ALTER COLUMNFAMILY listings add
pincodes_served set<int>;
```

Secondary Index

Query with Collections

```
cassandra@cqlsh:catalog>ALTER COLUMNFAMILY listings add
pincodes_served set<int>;
```

Add data

```
cassandra@cqlsh:catalog>update listings set pincodes_served =
pincodes_served+{560034,560041} where sellerid = 'Fab' and
skuid = 'SKU11' and  stock = 1 and ssp = 13000;
```

```
cassandra@cqlsh:catalog>update listings set pincodes_served =
pincodes_served+{560034,560041} where sellerid = 'Fab' and skuid = 'SKU1'
and  stock = 1 and ssp = 5000;
```

Secondary Index

Query with Collections

Create index on pincodes_served

```
cassandra@cqlsh:catalog> create index on listings(pincodes_served);
```

Secondary Index Query with Collections

```
cassandra@cqlsh:catalog> create index on listings(pincodes_served);
```

```
cassandra@cqlsh:catalog> select * from listings where pincodes_served contains 560034;
```

Notice the contains operation on the indexed column

Secondary Index Query with Collections

```
cassandra@cqlsh:catalog> select * from listings where pincodes_served contains 5600
34;

 sellerid | skuid | stock | ssp   | listingid        | mrp   | pincodes_served   | pr
oductid | sla | title
----------+-------+-------+-------+------------------+-------+-------------------+---
--------+-----+---------------------
      Fab |  SKU1 |    12 |  5000 |    LISTINGSOFA1 |  6000 | {560034, 560041} |
  SOFA1 |   5 | Urban Living Derby
      Fab | SKU11 |     1 | 13000 | LISTINGFABSOFA2 | 15000 | {560034, 560041} |
  SOFA2 |   3 |     Urban 2 seater

(2 rows)
```

IT WORKS!

Similarly for maps, we can use the CONTAINS KEY keyword

Index Key Restrictions

With index, either all the columns of the partition key should be restricted in the query or none

We cannot use the IN predicate on indexed column

Range operators >,<>=,<= are not supported

Use of clustering columns with index is not allowed

ORDER BY is not supported

CONTAINS, CONTAINS KEY operation on collections is supported

Secondary Index

Let's restrict indexed columns with range operator

```
cassandra@cqlsh:catalog> select * from listings where
productid > 'SOFA';
```

InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"

ALLOW FILTERING will enable the query to scan entire index

Secondary Index

Lets restrict indexed columns with range operator

```
cassandra@cqlsh:catalog> select * from listings where
productid > 'SOFA';
```

InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"

query is valid query, but its performance will be poor

Secondary Index

Lets restrict indexed columns with range operator

Let's run this query with ALLOW FILTERING

```
cassandra@cqlsh:catalog>select * from listings where
productid > 'SOFA' ALLOW FILTERING;
```

add the command at the end
of your query to allow filtering

Secondary Index

Lets restrict indexed columns with range operator

Let's run this query with ALLOW FILTERING

```
cassandra@cqlsh:catalog>select * from listings where
productid > 'SOFA' ALLOW FILTERING;
```

Now cassandra will first find and load all the rows in listings

And then filter out the rows where sellerId is lexically <= SOFA

Secondary Index

Lets restrict indexed columns with range operator

```
cassandra@cqlsh:catalog>select * from listings where
productid > 'SOFA' ALLOW FILTERING;
```

```
 sellerid | skuid | stock | ssp   | listingid       | mrp   | productid | sla | tit
le
----------+-------+-------+-------+-----------------+-------+-----------+-----+----
-----------------
      Fab |  SKU1 |    12 |  5000 |    LISTINGSOFA1 |  6000 |     SOFA1 |   5 |   U
rban Living Derby
    Decor |  SKU2 |     0 | 12000 |    LISTINGSOFA2 | 15000 |     SOFA2 |   3 | Urb
an Decor 2 seater
      Fab | SKU11 |     1 | 13000 | LISTINGFABSOFA2 | 15000 |     SOFA2 |   3 |
   Urban 2 seater

(3 rows)
```