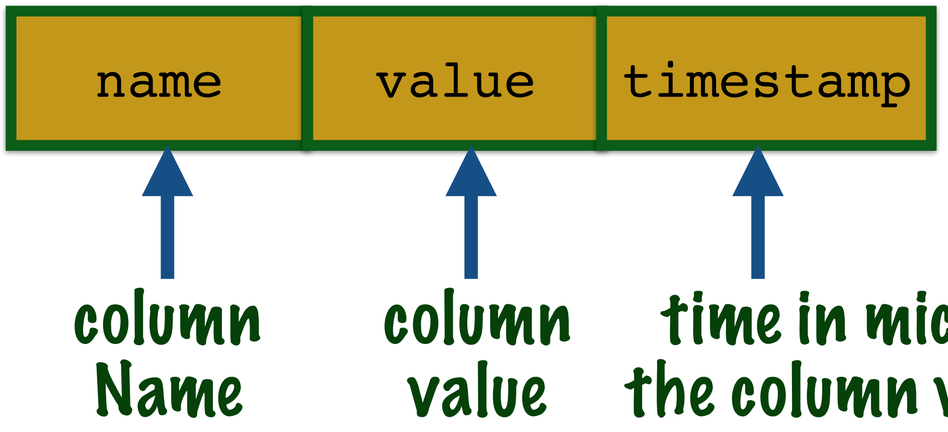
Example 7:
cqlsh
insert data

insert data

a collection of column-value pairs
form a row in cassandra

column consists of 3 parts



Data is stored as byte
arrays in cassandra

Let's add rows to product

```
cassandra@cqlsh:catalog> insert into product(productid, brand, modelid) values('MOB1
', 'Samsung', 'GalaxyS6');
```

command to add row to the
product column family

Let's add rows to product

```
cassandra@cqlsh:catalog> insert into product(productid, brand, modelid) values('MOB1
', 'Samsung', 'GalaxyS6');
```

columns in product
for which we want
to add data

Let's add rows to product

```
cassandra@cqlsh:catalog> insert into product(productid, brand, modelid) values('MOB1
', 'Samsung', 'GalaxyS6');
```

values for the respective columns

if client doesn't pass the timestamp then the timestamp at the server at the time of creation of column will we used

Let's add rows to product

adding more rows..

```
cassandra@cqlsh:catalog> insert into product(productid, title, publisher) values('BO
K1', 'The Kite Runner', 'Riverhead Books');
cassandra@cqlsh:catalog> insert into product(productid, title, breadth, length) valu
es('POST2', 'led zeppelin', 22, 36) USING TIMESTAMP 1468580580000;
```

command to pass
the timestamp
with the row data

We have provided only those columns
for which the product has data

Let's add rows to product

Time-To-Live(TTL) with the column value

column values are automatically marked as deleted
after the requested length of time has expired

```
cassandra@cqlsh:catalog> insert into product(productid, title, breadth, length)
values('POST2', 'adele', 22, 36 ) USING TTL 86400;
```

the entire row will be marked as
deleted after 86400 seconds (1 day)

Let's add rows to product

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

 productid | brand    | breadth | camera | height | keyfeatures | length | modelid  |
 publisher        | title
-----------+----------+---------+--------+--------+-------------+--------+----------+
-----------------+-----------------
      POST1 |     null |      22 |   null |   null |        null |     36 |     null |
             null |        hard work
       BOK1 |     null |    null |   null |   null |        null |   null |     null |
 Riverhead Books | The Kite Runner
       MOB1 | Samsung |    null |   null |   null |        null |   null | GalaxyS6 |
             null |             null

(3 rows)
```

The query and the output representation of the query result is very similar to sql

internally, the column value pair is not created if data is not present for the column

empty column values are represented as null to make the query output readable

This is not how data is actually stored within Cassandra!

This is not how data is actually stored within Cassandra!

Cassandra, internally, follows the Map<row key, SortedMap<key, value>> layout - a columnar layout

Let's add rows to product

Show me the data

cql supports '=' and 'IN' operations on primary keys

```
cassandra@cqlsh:catalog> select * from product where productid = 'POST1';
```

= operation

return data for the products with productid = 'POST1'

Let's add rows to product

Show me the data

cql supports '=' and 'IN' operations on primary keys

```
cassandra@cqlsh:catalog> select * from product where productid = 'POST1';

 productid | brand | breadth | camera | height | keyfeatures | length | modelid | pu
blisher | title
-----------+-------+---------+--------+--------+-------------+--------+---------+---
--------+-----------
     POST1 |  null |      22 |   null |   null |        null |     36 |    null |
   null | hard work

(1 rows)
```

Let's add rows to product

Show me the data

IN operation

```
cassandra@cqlsh:catalog> select productid, title, modelid from product where product
id IN ( 'POST1', 'MOB1');
```

return data for the products with
productid = 'POST1' or 'MOB1'

Let's add rows to product

Show me the data

IN operation

```
cassandra@cqlsh:catalog> select productid, title, modelid from product where product
id IN ( 'POST1', 'MOB1');

 productid | title      | modelid
-----------+------------+-----------
      MOB1 |       null | GalaxyS6
     POST1 | hard work |       null

(2 rows)
```

Let's add rows to product

Show me the data

We will revisit conditional queries
in more detail later

Example 8:

cqlsh
advanced data types
(collection , counter)

collection data types

set	collection of one or more distinct elements
list	collection of one or more ordered elements
map	data of key->value pairs

collections cannot be nested
set<set<text>> is not supported

can associate TTL with
individual elements in a collection

collection data types

set

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD keyfeatures set<text>;
```

keyfeatures stored as set of type text

collection data types

set

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD keyfeatures set<text>;
```

a set doesn't save duplicate entries.

```
cassandra@cqlsh:catalog> insert into product(productid, title, brand, keyfeatures)
values('COM1', 'Acer One', 'Acer',{'detachable keyboard','multitouch display'});
```

data is enclosed by curly braces{}

collection data types

list

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD service_type list<text>
```

list of type text

service_type is a sequence of steps to be followed for servicing

the order of the individual steps is important

So let's save the data as a list

collection data types

list

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD service_type list<text>;
```

```
cassandra@cqlsh:catalog> insert into product(productid, title, brand, service_type)
values('SOFA1', 'Urban Living Derby Sofa', 'Urban Living', ['Needs to Call Seller',
'Service Engineer will Come to the Site']);
```

the column values will be returned in
the order in which they are added

collection data types

map

to store data of key->value form

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD camera map<text,text>;
```

map data type

there can be 2 cameras
in a phone/tablet
front
rear

collection data types

map

to store data of key->value form

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD camera map<text,text>;
```

key is of type text
value is of type text

```
cassandra@cqlsh:catalog> insert into product(productid, title, brand, camera) values
('COM1', 'Acer One', 'Acer', {'front':'VGA', 'rear':'2MP'} ) ;
```

front and rear camera configurations saved in a map

COUNTER

a special column for keeping count of an event

eg counter for number of page views

Its just a number
Why don't we use an integer instead?

COUNTER

Its just a number
Why don't we use an integer instead?

for consistent behaviour using int data type we need to:

1. acquirelock on the id
2. read the counter
3. increment
4. release the lock

making it a serial operation

A Counter performs all these steps atomically

COUNTER

But there is no free lunch

Few restrictions when using counter

counter has to be stored in dedicated columnfamily

Create a new column family with primary key and only counter column

we cannot create an index on the counter column

we will see indexes later

COUNTER

creating a counter columnfamily

```
cassandra@cqlsh:catalog> create columnfamily productviewcount(productid text primar
y key, viewcount COUNTER);
```

we have only 2 columns here.
counter and primary key

Lets add data to the counter..

COUNTER

Updating counters

update command is used for loading data into counter as well as updating the counter

COUNTER

Updating counters

```
cassandra@cqlsh:catalog> update productviewcount set viewcount = viewcount + 1 wher
e productid = 'COM1';
```

update command is used for both inserting a new counter and updating an existing counter

COUNTER

Updating counters

```
cassandra@cqlsh:catalog> update productviewcount set viewcount = viewcount + 1 wher
e productid = 'COM1';
```

viewcount is our column with counter data type

we increment the counter by 1

COUNTER

Updating counters

```
cassandra@cqlsh:catalog> update productviewcount set viewcount = viewcount + 1 wher
e productid = 'COM1';
```

if COM1 doesn't exist in productviewcount then a new row will be created for COM1

viewcount value will be set according to the data given with the set operator

COUNTER

Updating counters

Let's verify the data

```
cassandra@cqlsh:catalog> select * from productviewcount where productid = 'COM1';

 productid | viewcount
-----------+-----------
      COM1 |         1

(1 rows)
```

Example 9:
cqlsh
simple update

Updating columns with basic data type

= operator

```
cassandra@cqlsh:catalog> UPDATE product set modelid='S6' where productid = 'MOB1';
```

update product
column family

data is always updated in append mode

Updating columns with basic data type

= operator

```
cassandra@cqlsh:catalog> UPDATE product set modelid='S6' where productid = 'MOB1';
```

new value for column modelid

Updating columns with basic data type

= operator

```
cassandra@cqlsh:catalog> UPDATE product set modelid='S6' where productid = 'MOB1';
```

condition on primary key

all columns that make up the primary key must be in the where clause

Updating columns with basic data type

IN operator

```
cassandra@cqlsh:catalog> update product set breadth=18, length=25, height=2 where pr
oductid in ('COM1', 'COM2');
```

we are updating multiple columns with columns separated by comma

Updating columns with basic data type

IN operator

```
cassandra@cqlsh:catalog> update product set breadth=18, length=25, height=2 where pr
oductid in ('COM1', 'COM2');
```

update data for columns in productid COM1, COM2

again all columns that compose the primary key must be in the where clause

Updating columns with collection data type

LIST

add element

```
set <column_name>=<column_name>+[new value]
```

remove element

```
set <column_name>=<column_name> - [exact existing value]
```

replace element

```
set <column_name>[index] ='new value'
```

index starts from 0

Updating columns with collection data type

LIST

add element

```
set <column_name>=<column_name>+[new value]
```

remove element

```
set <column_name>=<column_name> - [exact existing value]
```

replace element

```
set <column_name>[index] ='new value'
index starts from 0
```

Updating columns with collection data type

LIST

add element

```
set <column_name>=<column_name>+[new value]
```

remove element

```
set <column_name>=<column_name> - [exact existing value]
```

replace element

```
set <column_name>[index] ='new value'
index starts from 0
```

Updating columns with collection data type

LIST

add element

```
set <column_name>=<column_name>+[new value]
```

remove element

```
set <column_name>=<column_name> - [exact existing value]
```

replace element

```
set <column_name>[index] ='new value'
```

index starts from 0

Updating columns with collection data type

LIST

add element

```
cassandra@cqlsh:catalog> update product set service_type = service_type + ['Service
engineer will inspect the damages'] where productid='SOFA1';
```

update service_type column by adding a new element

Updating columns with collection data type

LIST

add element

```
cassandra@cqlsh:catalog> update product set service_type = service_type + ['Service
engineer will inspect the damages'] where productid='SOFA1';
```

update for only 1 product

Updating columns with collection data type

LIST

Lets verify the operation with select

Updating columns with collection data type

LIST

add element

```
 productid | brand         | breadth | camera | height | keyfeatures | length | model
id | publisher | service_type
                                                                                  | title
-----------+---------------+---------+--------+--------+-------------+--------+------
---+-----------+-----------------------------------------------------------------------
-----------------------------------------------+--------------------------
      SOFA1 | Urban Living |    null |   null |   null |        null |   null |    nu
ll |      null | ['Needs to Call Seller', 'Service Engineer will Come to the Site',
'Service engineer will inspect the damages'] | Urban Living Derby Sofa

(1 rows)
```

Updating columns with collection data type

LIST

remove element

```
cassandra@cqlsh:catalog> update product set service_type = service_type - ['Needs to
 Call Seller'] where productid = 'SOFA1';
```

update service_type by removing an element

Updating columns with collection data type

LIST

remove element

Lets verify the operation with select

Updating columns with collection data type

LIST

remove element

```
cassandra@cqlsh:catalog> select * from product where productid='SOFA1';

 productid | brand         | breadth | camera | height | keyfeatures | length | model
id | publisher | service_type
                      | title
-----------+---------------+---------+--------+--------+-------------+--------+------
---+-----------+-------------------------------------------------------------------------
--------------------+-------------------------

     SOFA1 | Urban Living |    null |   null |   null |        null |   null |    nu
ll |      null | ['Service Engineer will Come to the Site', 'Service engineer will i
nspect the damages'] | Urban Living Derby Sofa

(1 rows)
```

'Needs to Call Seller' value is removed

Updating columns with collection data type

LIST

replace element

```
cassandra@cqlsh:catalog> select * from product where productid='SOFA1';

 productid | brand         | breadth | camera | height | keyfeatures | length | model
id | publisher | service_type
                       | title
-----------+---------------+---------+--------+--------+-------------+--------+------
---+-----------+-------------------------------------------------------------------------
--------------------+--------------------------------------------------------------------

     SOFA1 | Urban Living |    null |   null |   null |        null |   null |    nu
ll |      null | ['Service Engineer will Come to the Site', 'Service engineer will i
nspect the damages'] | Urban Living Derby Sofa

(1 rows)
```

index 0

index 1

Let's update the value at index 1

Since list stores the data in order , we can access the elements by using index

Updating columns with collection data type

LIST

replace element

```
cassandra@cqlsh:catalog> select * from product where productid='SOFA1';

 productid | brand         | breadth | camera | height | keyfeatures | length | model
id | publisher | service_type
                      | title
-----------+---------------+---------+--------+--------+-------------+--------+------
---+-----------+-----------------------------------------------------------------------
--------------------+-------------------------
     SOFA1 | Urban Living |    null |   null |   null |        null |   null |    nu
ll |      null | ['Service Engineer will Come to the Site', 'Service engineer will i
nspect the damages'] | Urban Living Derby Sofa

(1 rows)
```

new value for index 1

```
cassandra@cqlsh:catalog> update product set service_type[1]='service engineer will c
all for appointment' where productid='SOFA1';
```

Updating columns with collection data type

LIST

replace element

Lets verify the operation with select

Updating columns with collection data type

LIST

replace element

```
cassandra@cqlsh:catalog> select * from product where productid='SOFA1';

 productid | brand         | breadth | camera | height | keyfeatures | length | model
id | publisher | service_type
                         | title
-----------+---------------+---------+--------+--------+-------------+--------+------
---+-----------+-------------------------------------------------------------------------
---------------------+-------------------------

     SOFA1 | Urban Living |    null |   null |   null |        null |   null |     nu
ll |      null | ['Service Engineer will Come to the Site', 'service engineer will c
all for appointment'] | Urban Living Derby Sofa

(1 rows)
```

new value for index 1

Updating columns with collection data type

SET

add element

```
set <column_name>=<column_name>+{new value}
```

remove element

```
set <column_name>=<column_name> - {exact existing value}
```

replace element not supported for set data type

Updating columns with collection data type

SET

add element

```
set <column_name>=<column_name>+{new value}
```

remove element

```
set <column_name>=<column_name> - {exact existing value}
```

replace element not supported for set data type

Updating columns with collection data type

SET

add element

```
set <column_name>=<column_name>+{new value}
```

remove element

```
set <column_name>=<column_name> - {exact existing value}
```

replace element not supported for set data type

Updating columns with collection data type

SET

add element

```
set <column_name>=<column_name>+{new value}
```

remove element

```
set <column_name>=<column_name> - {exact existing value}
```

replace element not supported for set data type

Updating columns with collection data type

SET

add element

```
cassandra@cqlsh:catalog> update  product USING TTL 86400 set keyfeatures = keyfeatur
es + {'FLAT 10% off for 1 day'} where productid in ('COM1', 'COM2');
```

We update the column using TTL of value 86400s

the entry will be marked to be deleted after 1 day

Updating columns with collection data type

SET

add element

```
cassandra@cqlsh:catalog> update  product USING TTL 86400 set keyfeatures = keyfeatur
es + {'FLAT 10% off for 1 day'} where productid in ('COM1', 'COM2');
```

Update keyfeatures by adding a new set element

Updating columns with collection data type

SET

add element

```
cassandra@cqlsh:catalog> update  product USING TTL 86400 set keyfeatures = keyfeatur
es + {'FLAT 10% off for 1 day'} where productid in ('COM1', 'COM2');
```

Update columns for products COM1 and COM2

Updating columns with collection data type

SET

add element

Lets verify the operation with select

Updating columns with collection data type

SET

add element

```
cassandra@cqlsh:catalog> select * from product where productid in ('COM1', 'COM2');

 productid | brand | breadth | camera                                                | height | keyfeature
s                                                                                  | length | modelid | p
ublisher | service_type | title
-----------+-------+---------+-------------------------------------------------------+--------+-----------
-----------------------------------------------------------------------------------+--------+---------+--
--------+--------------+----------
      COM1 |  Acer |      18 | {'front': '1MP', 'rear': '2MP'} |      2 | {'FLAT 10%
 off for 1 day', 'detachable keyboard', 'multitouch display'} |     25 |    null |
    null |         null | Acer One
      COM2 |  Acer |      18 |                                                  null |      2 | {'FLAT 10%
 off for 1 day', 'detachable keyboard', 'multitouch display'} |     25 |    null |
    null |         null | Acer One

(2 rows)
```

Updating columns with collection data type

SET

add element

Note that unlike list, in set the new element is added at the beginning and not at the end

```
cassandra@cqlsh:catalog> select * from product where productid in ('COM1', 'COM2');

 productid | brand | breadth | camera                                  | height | keyfeature
s                                                                    | length | modelid | p
ublisher | service_type | title
-----------+-------+---------+-----------------------------------------+--------+-----------
---------------------------------------------------------------------+--------+---------+---
---------+--------------+-----------
      COM1 |  Acer |      18 | {'front': '1MP', 'rear': '2MP'} |      2 | {'FLAT 10%
 off for 1 day', 'detachable keyboard', 'multitouch display'} |     25 |     null |
     null |         null | Acer One
      COM2 |  Acer |      18 |                                    null |      2 | {'FLAT 10%
 off for 1 day', 'detachable keyboard', 'multitouch display'} |     25 |     null |
     null |         null | Acer One

(2 rows)
```

Updating columns with collection data type

SET

remove element

```
cassandra@cqlsh:catalog> update product set keyfeatures = keyfeatures - {'detachable
 keyboard'} where productid IN ('COM1', 'COM2');
```

update key features column by removing an existing element from it

Note that this query will have no effect on the column data if after "-" operator you provide a value which doesn't exist in the keyfeatures column

Updating columns with collection data type

SET

remove element

```
cassandra@cqlsh:catalog> update product set keyfeatures = keyfeatures - {'detachable
 keyboard'} where productid IN ('COM1', 'COM2');
```

for columns pointed by
primary keys COM1, COM2

Updating columns with collection data type

SET

remove element

Lets verify the operation with select

Updating columns with collection data type

SET

```
cassandra@cqlsh:catalog> select * from product where productid in ('COM1', 'COM2');

 productid | brand | breadth | camera                                | height | keyfeature
s                                | length | modelid | publisher | service_type | title
-----------+-------+---------+---------------------------------------+--------+----------
-------------------------------+--------+---------+-----------+--------------+--------
--
      COM1 |  Acer |    null | {'front': 'VGA', 'rear': '2MP'} |   null | {'8 hr Bat
tery', 'multitouch display'} |   null |    null |      null |         null | Acer On
e
      COM2 |  Acer |    null |                                    null |   null | {'8 hr Bat
tery', 'multitouch display'} |   null |    null |      null |         null | Acer On
e

(2 rows)
```

'detachable keypad' value is removed

Updating columns with collection data type

MAP

add element

```
set <column_name>=<column_name> + {key:value}
```

remove element

delete command is used to remove an entry from map

```
delete <column_name>['key'] from <columnfamily>;
```

replace element

```
set <column_name>['key'] ='new value'
```

Updating columns with collection data type

MAP

add element

```
set <column_name>=<column_name> + {key:value}
```

remove element

delete command is used to remove an entry from map

```
delete <column_name>['key'] from <columnfamily>;
```

replace element

```
set <column_name>['key'] ='new value'
```

Updating columns with collection data type

MAP

add element

```
set <column_name>=<column_name> + {key:value}
```

remove element

delete command is used to remove an entry from map

```
delete <column_name>['key'] from <columnfamily>;
```

replace element

```
set <column_name>['key'] ='new value'
```

Updating columns with collection data type

MAP

add element

```
set <column_name>=<column_name> + {key:value}
```

remove element

delete command is used to remove an entry from map

```
delete <column_name>['key'] from <columnfamily>;
```

replace element

```
set <column_name>['key'] ='new value'
```

Updating columns with collection data type

MAP

add element

```
cassandra@cqlsh:catalog> update product set camera = camera + {'frontVideo':'1MP'} w
here productid = 'COM1' ;
```

update the
column camera by adding a
new key:value pair

Updating columns with collection data type

MAP

add element

```
cassandra@cqlsh:catalog> update product set camera = camera + {'frontVideo':'1MP'} w
here productid = 'COM1' ;
```

we are updating the column camera for product COM1

Updating columns with collection data type

MAP

add element

Lets verify the operation with select

Updating columns with collection data type

MAP

add element

```
cassandra@cqlsh:catalog> select * from product where productid = 'COM1';

 productid | brand | breadth | camera
| height | keyfeatures                                  | length | modelid | publisher | s
ervice_type | title
-----------+-------+---------+-----------------------------------------------------
+--------+-----------------------------------+--------+---------+-----------+--
------------+----------
      COM1 |  Acer |    null | {'front': 'VGA', 'frontVideo': '1MP', 'rear': '2MP'}
|   null | {'8 hr Battery', 'multitouch display'} |   null |    null |      null |
       null | Acer One

(1 rows)
```

Note that data is sorted based on the value of the key

Updating columns with collection data type

MAP

remove element

```
cassandra@cqlsh:catalog> delete camera['frontVideo'] from product where productid =
'COM1';
```

delete keyword is used for removing data.

delete command is also used for deleting rows from column family

Updating columns with collection data type

MAP

remove element

```
cassandra@cqlsh:catalog> delete camera['frontVideo'] from product where productid =
'COM1';
```

delete the key:value pair for "frontVideo"
key in column camera

Updating columns with collection data type

MAP

remove element

```
cassandra@cqlsh:catalog> delete camera['frontVideo'] from product where productid =
'COM1';
```

delete the data from product column family for product COM1

Updating columns with collection data type

MAP

remove element

Lets verify the operation with select

Updating columns with collection data type

MAP

remove element

```
cassandra@cqlsh:catalog> select * from product where productid = 'COM1';

 productid | brand | breadth | camera                                  | height | keyfeature
s                                | length | modelid | publisher | service_type | title
-----------+-------+---------+-----------------------------------------+--------+----------
--------------------------------+--------+---------+-----------+--------------+--------
--
      COM1 |  Acer |    null | {'front': 'VGA', 'rear': '2MP'} |   null | {'8 hr Bat
tery', 'multitouch display'} |   null |    null |      null |         null | Acer On
e

(1 rows)
```

Updating columns with collection data type

MAP

replace element

```
cassandra@cqlsh:catalog> update product set camera['front'] = '1MP' where productid
= 'COM1' ;
```

in column camera, change the value of the key "front" to "1MP"

Updating columns with collection data type

MAP

replace element

Lets verify the operation with select

Updating columns with collection data type

MAP

replace element

```
cassandra@cqlsh:catalog> select * from product where productid = 'COM1';

 productid | brand | breadth | camera                                  | height | keyfeature
s                                | length | modelid | publisher | service_type | title
-----------+-------+---------+-----------------------------------------+--------+----------
-------------------------------+--------+---------+-----------+--------------+--------
--
      COM1 |  Acer |    null | {'front': '1MP', 'rear': '2MP'} |   null | {'8 hr Bat
tery', 'multitouch display'} |   null |   null |      null |         null | Acer On
e

(1 rows)
```