**E-commerce Product Display Dashboard Scenario**

|  |
| --- |
| **Shopping Cart** |

|  |
| --- |
| **Order Details** |

Entity Used in E-commerce Product Display

|  |
| --- |
| **User** |

|  |
| --- |
| **User Comments** |

|  |
| --- |
| **Product** |

|  |
| --- |
| **Category** |

|  |
| --- |
| **Offers and Deals** |

|  |
| --- |
| **User Activity** |

Product Category

User and Cart Details

Search Product By Name

Best Deals and Offers

Top selling item and recommended product

Recently Viewed Item

|  |
| --- |
| User\_Id (Partition Key)  Cart\_Id (Cluster)  Cart\_Name text  First Name text  Last Name text  Type text  Address UDTs  cart\_item\_list List |

**Creating Column Families and Composite Partition Keys**

**Product\_by\_Cat Products Users\_and\_Carts**

|  |
| --- |
| Cat\_Name(Partition)  Cat\_Id  Prod\_List map<varchar,varchar> |

|  |
| --- |
| Prod\_Name(Partition)  Prod\_Id(Cluster)  Prod\_Price decimal  Prod\_Desc text |

**Deals\_and\_Offers Products\_by\_Recomands Recently\_Viewed\_Items**

|  |
| --- |
| Offer\_Id (Partition)  Cat\_Id(Partiton)  Discount\_Percent(Cluster)  Prod\_Id  Validity timeuuid |

|  |
| --- |
| Prod\_Name  Ratings(partition)  Discounts(cluster)  Prod\_Desc |

|  |
| --- |
| User\_Id(Partiton)  Prod\_Name(Cluster)  Viewed Counter |

**Table DDLs**

CREATE KEYSPACE krogerks\_ecom WITH replication = {'class': 'SimpleStrategy', 'replication\_factor': '3'} AND durable\_writes = true;

CREATE TABLE krogerks\_ecom.user (

userid text PRIMARY KEY,

first\_name text,

last\_name text,

type text

)

CREATE TABLE krogerks\_ecom.products (

prod\_name text,

prod\_id text,

prod\_desc text,

prod\_price decimal,

PRIMARY KEY (prod\_name, prod\_id)

)

CREATE INDEX prod\_desc ON krogerks\_ecom.products (prod\_desc);

CREATE INDEX prod\_price ON krogerks\_ecom.products (prod\_price);

CREATE TABLE krogerks\_ecom.prod\_by\_cat (

catname text,

catid text,

prod\_list map<text, text>,

PRIMARY KEY (catname, catid)

)

CREATE INDEX find\_by\_super\_column\_values ON krogerks\_ecom.prod\_by\_cat (values(prod\_list));

CREATE INDEX find\_by\_super\_column\_key ON krogerks\_ecom.prod\_by\_cat (keys(prod\_list));

CREATE INDEX find\_by\_super\_column\_entries ON krogerks\_ecom.prod\_by\_cat (entries(prod\_list));

CREATE TABLE krogerks\_ecom.status (

user\_id text,

status\_id text,

status text,

PRIMARY KEY (user\_id, status\_id)

)

CREATE TABLE krogerks\_ecom.users\_and\_carts (

user\_id text,

cart\_id text,

address address,

cart\_name text,

first\_name text,

last\_name text,

type text,

cart\_item\_list List<text>,

PRIMARY KEY (user\_id, cart\_id)

)

CREATE TABLE krogerks\_ecom.deals\_and\_offers (

offer\_id text,

prod\_id text,

validity timestamp,

discount\_percent decimal,

prod\_name text,

PRIMARY KEY (offer\_id, prod\_id, validity)

)

CREATE TABLE krogerks\_ecom.recently\_viewed\_items (

user\_id text,

prod\_name text,

viewed counter,

PRIMARY KEY (user\_id, prod\_name)

)

CREATE TABLE krogerks\_ecom.products\_by\_recommends (

rating float,

discount decimal,

prod\_desc text,

prod\_name text,

PRIMARY KEY (rating, discount)

)

**Data Setup Query**

**User**

insert into user(userid,first\_name,last\_name,type) values('U1','Praveen','Neelamegam','Prime');

insert into user(userid,first\_name,last\_name,type) values('U2','Rahul','Kumar','Non-Prime');

insert into user(userid,first\_name,last\_name,type) values('U3','Ajith','Kumar','Non-Prime');

insert into user(userid,first\_name,last\_name,type) values('U4','Sathish','Kumar','Prime');

**Prod\_by\_cat**

insert into prod\_by\_cat(catname,catid,prod\_list) values('Fashion','C1',{'P1':'Shirt','P2':'Jeans'});

insert into prod\_by\_cat(catname,catid,prod\_list) values('Fashion','C2',{'P11':'Tops','P2':'Jeans'});

insert into prod\_by\_cat(catname,catid,prod\_list) values('Electronic','C3',{'P21':'Mobile','P22':'Laptops'});

**Products**

insert into products(prod\_name,prod\_id,prod\_desc,prod\_price) values('Shirts','D1','Wrogn Shirts',600);

insert into products(prod\_name,prod\_id,prod\_desc,prod\_price) values('Shirts','D2','Arrow Shirts',500);

insert into products(prod\_name,prod\_id,prod\_desc,prod\_price) values('Mobile','M1','Nokia 1232',6000);

**Status**

insert into status(user\_id,status\_id,status) values('U1','S1','Active');

insert into status(user\_id,status\_id,status) values('U1','S2','InActive');

insert into status(user\_id,status\_id,status) values('U1','S3','Left');

**Users\_and\_carts**

insert into users\_and\_carts(user\_id,cart\_id,address,cart\_item\_list,cart\_name,first\_name,last\_name,type) values('U1','C1',{city:'Bangalore',state:'Karnataka',zipcode:560033},['Shirt','Jeans'],'MyCart','Praveen','Neelamegam','Prime');

insert into users\_and\_carts(user\_id,cart\_id,address,cart\_item\_list,cart\_name,first\_name,last\_name,type) values('U1','C2',{city:'Bangalore',state:'Karnataka',zipcode:560033},['Laptop','Mobile'],'MyCart2','Praveen','Neelamegam','Prime');

**Deals\_and\_offers**

insert into deals\_and\_offers(offer\_id,prod\_id,validity,discount\_percent,prod\_name) values('OFF2','P2','2019-07-24',40,'Lenova Laptop');

insert into deals\_and\_offers(offer\_id,prod\_id,validity,discount\_percent,prod\_name) values('OFF1','P1','2019-07-01',30,'HP Laptop');

**Recently\_viewed\_items**

update recently\_viewed\_items set viewed=viewed+1 where user\_id='U1' and prod\_name='Hp Laptop';

update recently\_viewed\_items set viewed=viewed+1 where user\_id='U1' and prod\_name='Lenova Laptop';

**Products\_by\_recommends**

insert into products\_by\_recommends(rating,discount,prod\_desc,prod\_name) values(1,40,'Laptops','Hp Laptop');

insert into products\_by\_recommends(rating,discount,prod\_desc,prod\_name) values(2,30,'Laptops','Lenova Laptop');

insert into products\_by\_recommends(rating,discount,prod\_desc,prod\_name) values(4,30,'Mobile','Nokia Mobile');

**Counter**

* Data type - Used to store 64 bit signed integer
* Changes incrementally (increment and decrement)
* values are changed using Update
* Need special dedicated table which can hold only primary key and counter columns(can have more than one counter column )
* cannot insert or assign values to counter column - default value is 0
* must be only non primary key, it cannot be indexed or deleted

table used : recently\_viewed\_items

create table recently\_viewed\_items(user\_id text,prod\_name text,viewed counter,primary key(user\_id,prod\_name));

update recently\_viewed\_items set viewed=viewed+1 where user\_id='U2' and prod\_name='Nokia Mobile';

Counter type allows concurrent increment/decrement on the value without neither requiring read-before-write nor locking

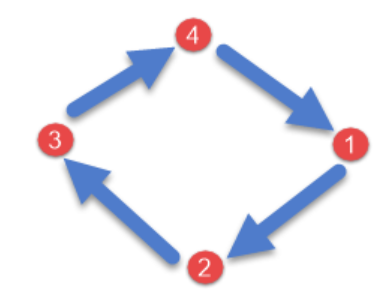
**Defining the replication factor**

Cassandra places replicas of data on different nodes based on these two factors

* Where to place next replica is determined by the Replication Strategy.
* While the total number of replicas placed on different nodes is determined by the Replication Factor.

**SimpleStrategy**

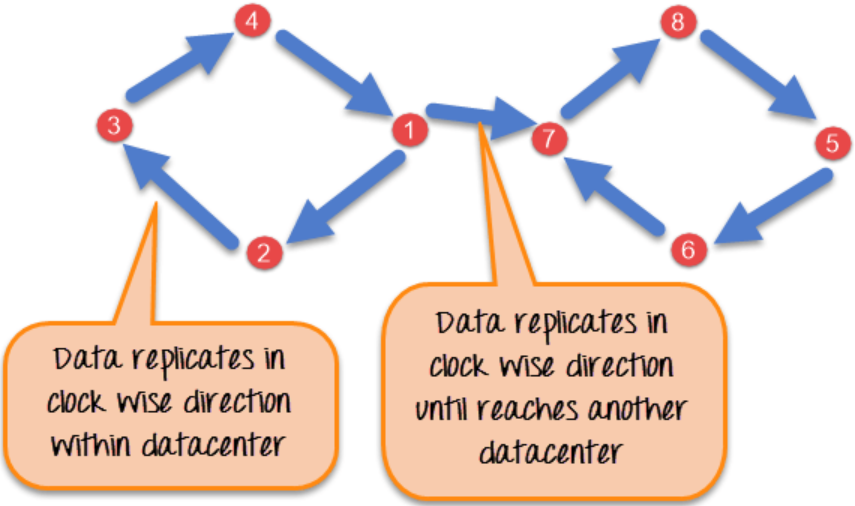
* It is used when you have just one data center.
* It places the first replica on the node selected by the partitioner. After that, remaining replicas are placed in clockwise direction in the Node ring



CREATE KEYSPACE krogerks\_ecom WITH replication = {'class':'SimpleStrategy', 'replication\_factor' : 3};

**NetworkTopologyStrategy**

* It is used when you have more than two data centers.
* In NetworkTopologyStrategy, replicas are set for each data center separately. NetworkTopologyStrategy places replicas in the clockwise direction in the ring until reaches the first node in another rack.
* This strategy tries to place replicas on different racks in the same data center. This is due to the reason that sometimes failure or problem can occur in the rack. Then replicas on other nodes can provide data.



CREATE KEYSPACE Kroger\_eCom

WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'datacenter1' : 3,'datacenter2' : 2 }

**TTL** (time-to-live)

* TTL is an optional expiration period
* TTL is not supported on counter columns and TTL value is defined in seconds
* Data expires after the data exceeds the TTL period and is then marked with a [tombstone](https://docs.datastax.com/en/glossary/doc/glossary/gloss_tombstone.html).
* Expired data continues to be available for read requests during the grace period
* To change the TTL of a specific column, you must re-insert the data with a new TTL
* Set TTL as 0, to switch off TTL on particular row in a column families.

Table : deals\_and\_offers

insert into deals\_and\_offers(offer\_id,prod\_id,validity,discount\_percent,prod\_name) values('OFF1','P1','2019-06-26',30.0,'HP Laptop') using ttl 3600;

Query to check TTL values : select prod\_name, TTL(prod\_name) from deals\_and\_offers;

**Row Caching and Key Caching**

Key Cache

* The key cache holds the location of keys in memory on a per-column family basis
* Key caching is enabled by default, at a level of 200,000 keys.
* It helps Cassandra know where the partition is located on disk, decreasing seek times

Row Cache

* Row cache holds the entire contents of the row in memory
* With row caching enabled, Cassandra will detect frequently accessed partitions and store rows of data into RAM to limit the cases where it needs to read from disk
* It is best used when you have a small subset of data to keep hot and you frequently need most or all of the columns returned

To use the row cache, you must also instruct Cassandra how much memory you wish to dedicate to the cache using the row\_cache\_size\_in\_mb setting in the cassandra.yaml config file

CREATE TABLE status (

user\_id text,

status\_id text,

status text,

PRIMARY KEY (user\_id,status\_id)) WITH caching = {'keys':'ALL', 'rows\_per\_partition':'2'};

insert into status(user\_id,status\_id,status) values('U1','S1','Active');

insert into status(user\_id,status\_id,status) values('U1','S2','InActive');

Testing row Caching

cqlsh> tracing on;

cqlsh> select \* from status where user\_id='U1' limit 2;

Row cache miss [ReadStage:41]

Row cache hit [ReadStage:55]

**Indexes**

* It provides means to access data in Cassandra using non-primary key fields other than the partition key
* An index indexes column values in a separate, hidden column family (table) from the one that contains the values being indexed
* The data of an index is local only, which means it will not be replicated to other nodes.
* For data query by indexed column, the requests has to be forwarded to all the nodes, waiting for all the responses, and then the results are merged and returned, so if you have many nodes, the query response slows down as more machines are added to the cluster

Do not use an index in these situations

* On high-cardinality columns because you then query a huge volume of records for a small number of results
* In tables that use a counter column
* To look for a row in a large partition unless narrowly queried

create index type on user(type);

select \* from user where type='Prime';

**Find all the category of a particular product based on product id from prod\_by\_cat;**

create index find\_by\_super\_column\_key on prod\_by\_cat (keys(prod\_list));

select \* from prod\_by\_cat where prod\_list contains key 'P1';

**Find all the category of a particular product based on product entries in Super Column**

create index find\_by\_super\_column\_entries on prod\_by\_cat (ENTRIES(prod\_list));

select \* from prod\_by\_cat where prod\_list['P13']='Mobile';

**Find all the category of a particular product based on product name in Super Column**

create index find\_by\_super\_column\_values on prod\_by\_cat (VALUES(prod\_list));

select \* from prod\_by\_cat where prod\_list contains 'Laptop';

**Consistency**

* Cap theorem - Cassandra falls under availability and partiton tolerance
* consistence level to be set for read and write by developer basd on RF
* CL=One - expects acknowledgement from any one replicated node
* CL=QUORUM - expects acknowledgements from 51% of nodes ((total no of nodes + 1)/2)
* CL=ALL - expects acknowledgements from all the nodes where data is replicated

**Strong Consistency**

* write CL=ALL and read CL=ONE
* write CL=QUORUM read CL=QUORUM which maintain availability best practice in production environments

**Eventually consistency**

* write CL=ONE and read CL=ONE

**Consistency Across Data Center**

* LOCAL\_QUORUM or LOCAL\_ONE or EACH\_QUORUM on particular data center

The default consistency level for any query is "ONE"

CQL -> CONSISTENCY QUORUM;

Queries Mostly Used in this scenario

Q1 -> Fetch all Products which has best deals and offers of the day

Q2 -> Fetch about Product details

Q3 -> Fetch about User details

Q4 -> Fetch all the similar category of Product based on the product you chosen

Q5 -> Fetch recent 10 user comments on chosen Product

Q6 -> Display product which is liked by the User

Q7 -> Product liked by user

Q8 ->User activities Over last 30 days

Q9 ->Query to find order details and status of logged in user

|  |
| --- |
| **Product**  Prod\_Id (Partition Key)  Cat\_Id (Cluster Key)  Prod\_Name text  Prod\_Price decimal  Description text  Prod\_Color map<,> |

Tables: User, Product, Offer\_and\_Deals, User\_Comments , Category, Shopping Cart, User Activity and

|  |
| --- |
| **User**  User\_Id (Primary Key)  First Name text  Last Name text  Type text  Address UDTs |

|  |
| --- |
| **Offers\_and\_Deals**  Offer\_Id (Partition Key)  Prod\_Id (Cluster Key)  Discount\_percent static  Validity timeuuid |

Orders

|  |
| --- |
| **Category**  Cat\_Id (Primary)  Prod\_List  map<varchar, varchar> |

|  |
| --- |
| **User Comments**  User\_Id (Partition)  Prod\_Id (Cluster)  First Name text  Last Name text  Comments text |

|  |
| --- |
| **Shopping Cart**  User\_Id(Partition)  Cart\_Name(Partition)  Item\_Id(Cluster)  User\_Name text  Item\_Name text  Price decimal  Item\_details text |

|  |
| --- |
| **User Activity**  User\_Id(Partition)  Interaction\_Time (Cluster)  Activity\_Code int  Details text |

Q1 -> **Product\_by\_Best\_Deals**

|  |
| --- |
| Offer\_Id (Partition)  Prod\_Id (cluster key)  Prod\_Name  Prod\_Price  Discount\_Percent  Validity (Cluster) |

Select Prod\_Name ,Discount\_Percent from Product\_by\_Best\_Deals where Offer\_Id=? and Validity>=SYSTEM\_DATE;

Q2->**Product\_by\_Cat**

Select Prod\_Id,Prod\_Name,Prod\_Price from Product where Prod\_Id=? and Cat\_Id=?

|  |
| --- |
| Cat\_Id (Partition Key)  Prod\_Id (Cluster Key)  Prod\_Name  Prod\_Price |

|  |
| --- |
| User\_Id (Primary)  First Name  Last Name  Address |

Q4->**User**

Select First\_Name,Last\_Name , Address from User where User\_id=?

Q5 -> **Product\_by\_Cat**

|  |
| --- |
| Cat\_Id (Partition Key)  Prod\_Id (Cluster Key)  Prod\_Name  Prod\_Price |

Select Prod\_Id,Prod\_Name,Prod\_Price from Category where Cat\_Id=? allow filtering;

Q6 -> **User\_Comments\_by\_Product**

|  |
| --- |
| Prod\_Id (Partition)  User\_Id (cluster)  First Name  Last Name  Comments |

Select Comments\_List from User\_Comments\_by\_Product

Where Prod\_Id=? allow filtering;

Q7 ->**Product\_liked\_by\_User**

Select Prod\_Name from Product\_liked\_by\_User where User\_Id=?

|  |
| --- |
| User\_Id (partition)  Prod\_Id(cluster)  Prod\_Name  Prod\_Price |

Q8 ->**User activity over last 30 days**

Insert query: insert into user\_activity(user\_id,user\_name,interation\_time,activity\_code,details)

values(‘U1’,(‘Praveen’,’Neelamegam’),toTimestamp(now()),’100’,’Login’)

using ttl 2592000; (expire after 30 days)

select activity\_code,details from where user\_id=? and interaction\_time=?;

Q9 ->**Query to find Order details and status by logged in user**

Select \* from orders where user\_id=? and order\_id=? and cat\_name=?

Q10 ->**Using Index Feature to find a user by their first name**

create index find\_by\_name on user (first\_name);

select \* from user where first\_name='Praveen';

Q11 ->**Find all the category of a particular product based on product id**

create index find\_by\_super\_column\_key on category (keys(products));

select \* from category where products contains key 'P1';

Q12 ->**Find all the category of a particular product based on product entries in Super Column**

create index find\_by\_super\_column\_entries on category (ENTRIES(products));

select \* from category where products['P1']='Mobile';

Q13 -> **Find all the category of a particular product based on product name in Super Column**

create index find\_by\_super\_column\_values on category (VALUES(products));

select \* from category where products contains 'Laptop';

FROZEN datatype is used to specify columns of binary strings that result from serializing either collections, tuples, or user-defined types.

Q14 ->**Query to alter table with new super key column with list of map type**

alter table prod\_by\_cat add prod\_colors map<text, frozen<list<varchar>>>;

select \* from prod\_by\_cat where prod\_colors contains ['Red','Yellow'];

Q15 ->**Query to create user defined data type**

create type customer\_details ( first\_name text,last\_name text, nationality text,address text);

Q16 ->**Query to alter product\_liked\_by\_user table to add customer details column and insert value for customer details UDTs**

alter table add cust\_details frozen<customer\_details>;

update product\_liked\_by\_user set cust\_details={first\_name:'Praveen',last\_name:'Neelamegam',nationality :'Indian',address:'asdfasd'} where user\_id='U1' and prod\_id='P1';

Q17 ->**Query to display user details who likes particular product**

create index on product\_liked\_by\_user (prod\_id); (Since prod\_id is non partition key)

select cust\_details from product\_liked\_by\_user where prod\_id='P1';

**Json and Tuple Query example**

Q18 ->**Query the table result in JSON format**

select json \* from prod\_by\_cat; Or select json prod\_colors from prod\_by\_cat;

Q19 ->**Query to insert json format values in the table**

insert into prod\_by\_cat json

'{ "cat\_id": "C2",

"prod\_id": "P1",

"prod\_colors":

{ "Vega":[

"Red",

"Blue"

],

"Gionee":[

"White",

"Balck"

]

},

"prod\_name": "Helmet",

"prod\_price": 400.00

}';

Q20 ->**Query to alter user table address field as tuple instead of type text and insert new row**

alter table user add address tuple<text,int,text>;

update user set address=('Banglore',560033,'India') where user\_id='U1';

insert into user(user\_id,first\_name,last\_name,type,address) values('U5','Megha','Verma’,’Prime',('Banglore',null,'India'));

Q21 ->**Query to filter on tuple column**

CREATE INDEX ON user (address);

select \* from user where address=('Banglore',560033,'India');

**Static**

Q22 ->**Query to make column static in prod\_by\_best\_deals table and insert new row of exiting partition key**

alter table prod\_by\_best\_deals add discount\_percent decimal static;

insert into prod\_by\_best\_deals(offer\_id,cat\_id,prod\_id,prod\_name,Validity) values('OFF3','C4','P4','AC',toTimestamp(now()));

Light Weight Transection and Optimistic locking concept

Q23 ->**Query to insert data with conditional write**

insert into user(user\_id,first\_name,last\_name,type,address) values('U1','Ramki','Neelamegam’,'Prime',('Banglore',560021,'India')) if not exists;

Q24 ->**Avoiding Update Query to insert new row in to the table (Optimistic locking)**

Inserts new row -> update user set type='Prime' where user\_id='U3';

ALTER TABLE "user" ADD "version" timeuuid;

UPDATE "user" SET "version" = NOW() where user\_id=’U1’

update user set type='Prime' where user\_id='U1' if version=42b90b40-8dae-11e9-8b45- 75a0e3bc8974;

Q25 ->**Query to manage concurrent update (Optimistic locking on concurrent update)**

UPDATE "user" SET "first\_name" = 'Megha’, "version" = NOW() WHERE "user\_id" = 'U3' IF "version" = 4ae442d0-8dae-11e9-8b45-75a0e3bc8974;