Let's develop a JAVA application for a

Miniature Catalog Management System (cms)

Miniature Catalog Management System

the project is divided into 3 modules

Product

Listing

Persistence

it also has a main class which sets up the database and performs all operations on it

Miniature Catalog Management System

Product

It contains the catalog attributes of the product. All attributes which define a product

Miniature Catalog Management System

Listing

This class contains the pricing and stock details of the products sold by seller

A seller may sell multiple products and multiple products may be sold by a seller

Miniature Catalog Management System

Persistence

It contains the connector to connect to cassandra cluster to perform operations

Miniature Catalog Management System

Main

This is the main executing class which uses the helper methods in the other classes to perform operations cms is a maven based project module

Maven is a build automation tool used to manage java projects

If you do not have maven installed on your machine please follow the installation video to set it up

To interact with cassandra, we use the datastax cassandra-driver

Let's define the dependency in pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>com.datastax.cassandra</groupId>
        <artifactId>cassandra-driver-core</artifactId>
        <version>3.0.2</version>
    </dependency>
</dependencies>
```

All cql operations have a corresponding method in the java-driver (another way to reference the cassandra-driver)

Let's begin with creating a keyspace

This is in Main.java

cql

```
cassandra@cqlsh> CREATE KEYSPACE cms WITH replication = {'class':
'SimpleStrategy', 'replication_factor': '3'}  AND durable_writes =
true;
```

Java

```
static void createKeyspace(String keyspace){
    String query = "CREATE KEYSPACE "+ keyspace+" WITH replication "
            + "= {'class':'SimpleStrategy', 'replication_factor':3};";
    Session session = Connector.getSession();
    session.execute(query);
    System.out.println("Keyspace created :"+keyspace);
}
```

It involves 3 steps

```
static void createKeyspace(String keyspace){
    String query = "CREATE KEYSPACE "+ keyspace+" WITH
replication “ + "= {'class':'SimpleStrategy',
'replication_factor':3};";
   Session session = Connector.getSession();
   session.execute(query);
  System.out.println("Keyspace created :"+keyspace);
}
```

1. defining the query

It involves 3 steps

1. defining the query

```
static void createKeyspace(String keyspace){
   String query = "CREATE KEYSPACE "+ keyspace+" WITH
replication “ + "= {'class':'SimpleStrategy',
'replication_factor':3};";
   Session session = Connector.getSession();
   session.execute(query);
  System.out.println("Keyspace created :"+keyspace);
 }
```

2. get the session object

It involves 3 steps

1. defining the query
2. get the session object

```
static void createKeyspace(String keyspace){
    String query = "CREATE KEYSPACE "+ keyspace+" WITH 
replication “ + "= {'class':'SimpleStrategy', 
'replication_factor':3};";
   Session session = Connector.getSession();
   session.execute(query);
  System.out.println("Keyspace created :"+keyspace);
}
```

3. executing the query

It involves 3 steps

1. defining the query
2. get the session object
3. executing the query

```
static void createKeyspace(String keyspace){
    String query = "CREATE KEYSPACE "+ keyspace+" WITH replication
"
            + "= {'class':'SimpleStrategy', 'replication_factor':
3};";
    Connector.getSession()
            .execute(query);
    System.out.println("Keyspace created :"+keyspace);
}
```

It involves 3 steps

2. get the session object

This is implemented in Connector.java

It involves 3 steps

2. get the session object

This is implemented in Connector.java

```
static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL,
MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL,
CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
            addContactPoint(host).
            withPort(port).
            withCredentials(userName, password).
            withPoolingOptions(poolingOptions).
            withClusterName(clusterName).
            build();

    session = cluster.connect();
    return session;
}
```

It involves 3 steps

2. get the session object

2.1 create a cluster object

```
static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL,
MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL,
CORE_CONNECTIONS);
   // Create a cluster object
    cluster = Cluster.builder().
                addContactPoint(host).
                withPort(port).
                withCredentials(userName, password).
                withPoolingOptions(poolingOptions).
                withClusterName(clusterName).
                build();

    session = cluster.connect();
    return session;
}
```

It involves 3 steps

2. get the session object

```
static void Session getSession() {
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL,
MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL,
CORE_CONNECTIONS);
```

2.1. create a cluster object

```
    // Create a cluster object
    cluster = Cluster.builder().
              addContactPoint(host).
              withPort(port).
              withCredentials(userName, password).
              withPoolingOptions(poolingOptions).
              withClusterName(clusterName).
              build();
```

2.2 connect to the cassandra cluster

```
     session = cluster.connect();
     return session;
```

1. defining the query

2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```
static void createKeyspace(String keyspace){
    String query = "CREATE KEYSPACE "+ keyspace+" WITH
replication “ + "= {'class':'SimpleStrategy',
'replication_factor':3};";
    Connector.getSession()
             .execute(query);
    System.out.println("Keyspace created :"+keyspace);
}
```

This is the same command that we executed on cqlsh

1. defining the query

2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```
Connector.getSession()
 static void Session getSession(){
     // We are configuring the connection pool
     PoolingOptions poolingOptions = new PoolingOptions();
     poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL,
 MAX_CONNECTIONS);
     poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL,
 CORE_CONNECTIONS);

     // Create a cluster object
     cluster = Cluster.builder().
             addContactPoint(host).
             withPort(port).
             withCredentials(userName, password).
             withPoolingOptions(poolingOptions).
             withClusterName(clusterName).
             build();

     session = cluster.connect();
     return session;
 }
```

1. defining the query

2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

session object contains

metadata about the keyspaces, column families in the cluster

holds connections to the cluster

```
Connector.getSession()
 static void Session getSession(){
     // We are configuring the connection pool
     PoolingOptions poolingOptions = new PoolingOptions();
     poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL,
 MAX_CONNECTIONS);
     poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL,
 CORE_CONNECTIONS);

     // Create a cluster object
     cluster = Cluster.builder().
             addContactPoint(host).
             withPort(port).
             withCredentials(userName, password).
             withPoolingOptions(poolingOptions).
             withClusterName(clusterName).
             build();

     session = cluster.connect();
     return session;
 }
```

1. defining the query
2. get the session object

2.1. create a cluster object

2.2. connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
              addContactPoint(host).
              withPort(port).
              withCredentials(userName, password).
              withPoolingOptions(poolingOptions).
              withClusterName(clusterName).
              build();

    session = cluster.connect();
    return session;
}
```

create a cluster object

1. defining the query

2. get the session object

2.1. create a cluster object

2.2. connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
                 addContactPoint(host).
                 withPort(port).
                 withCredentials(userName, password).
                 withPoolingOptions(poolingOptions).
                 withClusterName(clusterName).
                 build();

    session = cluster.connect();
    return session;
}
```

We use 9042 port

java-driver uses Binary Protocol to connect to the cluster

1. defining the query
2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
                addContactPoint(host).
                withPort(port).
                withCredentials(userName, password).
                withPoolingOptions(poolingOptions).
                withClusterName(clusterName).
                build();

    session = cluster.connect();
    return session;
}
```

the node and port
with which we use to
connect to the cluster

1. defining the query

2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
              addContactPoint(host).
              withPort(port).
              withCredentials(userName, password).
              withPoolingOptions(poolingOptions).
              withClusterName(clusterName).
              build();

    session = cluster.connect();
    return session;
}
```

we can add more than 1 host

In case 1 of the nodes is down, we can connect to the cluster using the other nodes

1. defining the query

2. get the session object

2.1. create a cluster object

2.2. connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
               addContactPoint(host).
               withPort(port).
               withCredentials(userName, password)
               withPoolingOptions(poolingOptions).
               withClusterName(clusterName).
               build();

    session = cluster.connect();
    return session;
}
```

we created easybuy using PasswordAuthenticator

We need to provide credentials

1. defining the query

2. get the session object

2.1. create a cluster object

2.2. connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
               addContactPoint(host).
               withPort(port).
               withCredentials(userName, password).
               withPoolingOptions(poolingOptions).
               withClusterName(clusterName).
               build();

    session = cluster.connect();
    return session;
}
```

we set the pooling options as per our application's requirement

1. defining the query

2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
     poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
     poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
              addContactPoint(host).
              withPort(port).
              withCredentials(userName, password).
              withPoolingOptions(poolingOptions).
              withClusterName(clusterName).
              build();

    session = cluster.connect();
    return session;
```

maxConnections is the maximum number of connections a host is allowed to make to the cluster

we have set it to 100

1. defining the query
2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
     poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
     poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
            addContactPoint(host).
            withPort(port).
            withCredentials(userName, password).
            withPoolingOptions(poolingOptions).
            withClusterName(clusterName).
            build();

    session = cluster.connect();
    return session;
 }
```

this is based on the number. of concurrent threads in our application that need to connect to cluster

1. defining the query

2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```java
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```java
Connector.getSession()
 static void Session getSession(){
     // We are configuring the connection pool
     PoolingOptions poolingOptions = new PoolingOptions();
      poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
      poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

     // Create a cluster object
     cluster = Cluster.builder().
               addContactPoint(host).
               withPort(port).
               withCredentials(userName, password).
               withPoolingOptions(poolingOptions).
               withClusterName(clusterName).
               build();

     session = cluster.connect();
     return session;
}
```

core connections are the minimum number of connections required by the application to start

we have set it to 25

1. defining the query
2. get the session object

2.1. create a cluster object

2.2. connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
               addContactPoint(host).
               withPort(port).
               withCredentials(userName, password).
               withPoolingOptions(poolingOptions).
               withClusterName(clusterName).
               build();

    session = cluster.connect();
    return session;
}
```

we set the pooling options as per our applications requirement

1. defining the query

2. get the session object

2.1. create a cluster object

2.2. connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
               addContactPoint(host).
               withPort(port).
               withCredentials(userName, password).
               withPoolingOptions(poolingOptions).
               withClusterName(clusterName).
               build();

    session = cluster.connect();
    return session;
}
```

name of the cluster we want to connect

1. defining the query

2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

Connector.getSession()

```
static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
            addContactPoint(host).
            withPort(port).
            withCredentials(userName, password).
            withPoolingOptions(poolingOptions).
            withClusterName(clusterName).
            build();

    session = cluster.connect();
    return session;
}
```

When we connect to the cluster, a session object is returned

1. defining the query

2. get the session object

2.1. create a cluster object

2.2. connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
    // We are configuring the connection pool
    PoolingOptions poolingOptions = new PoolingOptions();
    poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
    poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

    // Create a cluster object
    cluster = Cluster.builder().
            addContactPoint(host).
            withPort(port).
            withCredentials(userName, password).
            withPoolingOptions(poolingOptions).
            withClusterName(clusterName).
            build();

    session = cluster.connect();
    return session;
 }
```

session object contains

connection pools

metadata about the keyspaces, column families in the cluster

1. defining the query

2. get the session object

2.1. create a cluster object

2.2 connect to the cassadra cluster

3. executing the query

```
private static String host = "localhost";
private static int port = 9042;
private static String userName = "cassandra";
private static String password = "cassandra";
private static String clusterName = "easybuy";
private static int MAX_CONNECTIONS = 100;
private static int CORE_CONNECTIONS = 25;
```

```
Connector.getSession()
 static void Session getSession(){
     // We are configuring the connection pool
     PoolingOptions poolingOptions = new PoolingOptions();
     poolingOptions.setMaxConnectionsPerHost(HostDistance.LOCAL, MAX_CONNECTIONS);
     poolingOptions.setCoreConnectionsPerHost(HostDistance.LOCAL, CORE_CONNECTIONS);

     // Create a cluster object
     cluster = Cluster.builder().
             addContactPoint(host).
             withPort(port).
             withCredentials(userName, password).
             withPoolingOptions(poolingOptions).
             withClusterName(clusterName).
             build();

     session = cluster.connect();
     return session;
 }
```

session object is used to execute the queries on the cluster

1. defining the query

2. get the session object

    2.1. create a cluster object

    2.2 connect to the cassadra cluster

3. executing the query

```
static void createKeyspace(String keyspace){
    String query = "CREATE KEYSPACE "+ keyspace+" WITH replication "
            + "= {'class':'SimpleStrategy', 'replication_factor':3};";
    session = Connector.getSession();
    session.execute(query);
    System.out.println("Keyspace created :"+keyspace);
}
```

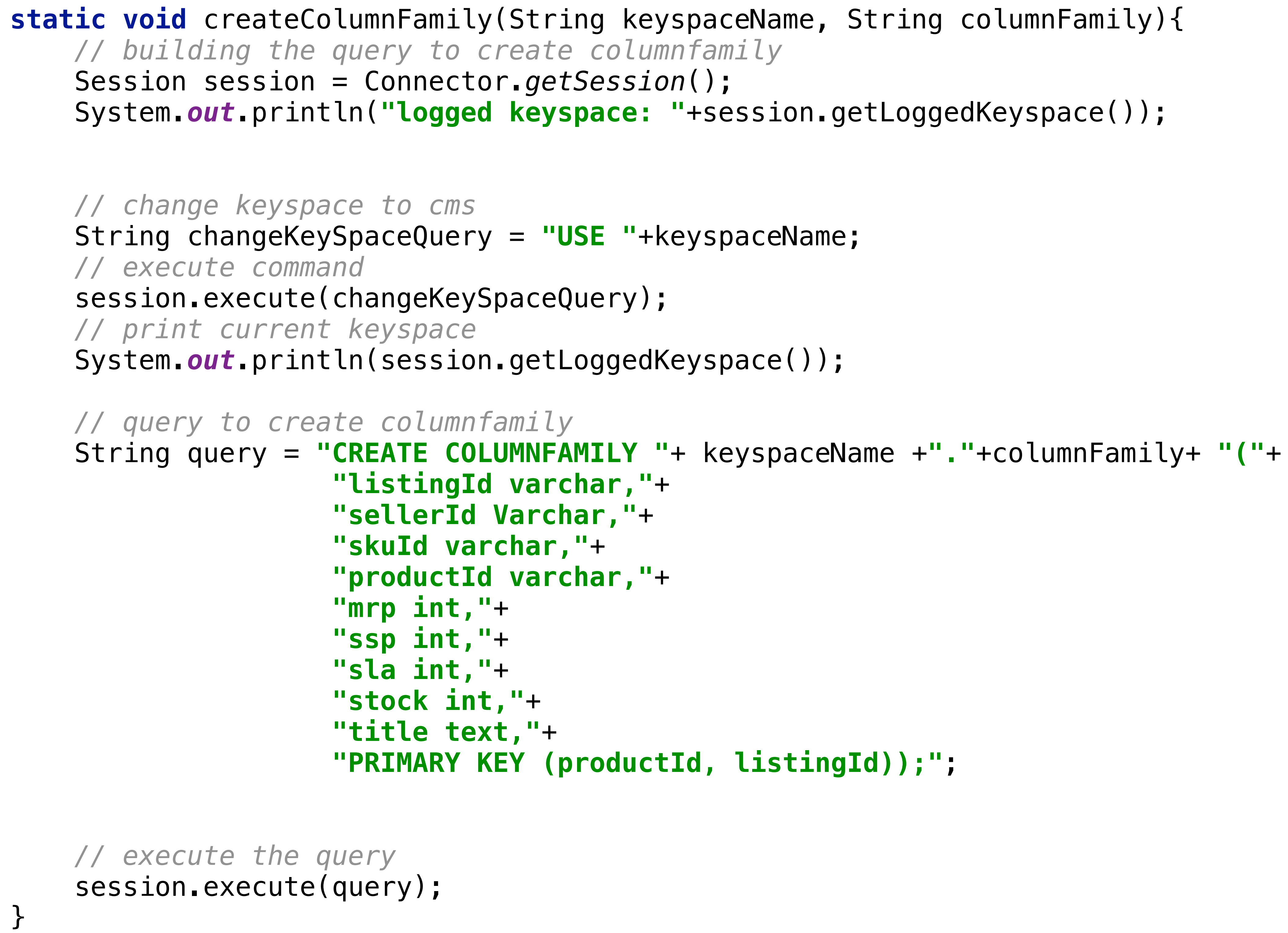
session object is used to execute
the queries to the cluster

Let's create a column family

cql

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

Java



```java
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();
    System.out.println("logged keyspace: "+session.getLoggedKeyspace());


    // change keyspace to cms
    String changeKeySpaceQuery = "USE "+keyspaceName;
    // execute command
    session.execute(changeKeySpaceQuery);
    // print current keyspace
    System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
    String query = "CREATE COLUMNFAMILY "+ keyspaceName +"."+columnFamily+ "("+
                    "listingId varchar,"+
                    "sellerId Varchar,"+
                    "skuId varchar,"+
                    "productId varchar,"+
                    "mrp int,"+
                    "ssp int,"+
                    "sla int,"+
                    "stock int,"+
                    "title text,"+
                    "PRIMARY KEY (productId, listingId));";


    // execute the query
    session.execute(query);
}
```

Java

```java
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();

    // change keyspace to cms
    String changeKeySpaceQuery = "USE "+keyspaceName;
    // execute command
    session.execute(changeKeySpaceQuery);
    // print current keyspace
    System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
    String query = "CREATE COLUMNFAMILY "+ keyspaceName +"."+columnFamily+ "("+
                    "listingId varchar,"+
                    "sellerId Varchar,"+
                    "skuId varchar,"+
                    "productId varchar,"+
                    "mrp int,"+
                    "ssp int,"+
                    "sla int,"+
                    "stock int,"+
                    "title text,"+
                    "PRIMARY KEY (productId, listingId));";

    // execute the query
    session.execute(query);
}
```

the query to create a columnfamily is defined

it is same as the one we used in cqlsh

```
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();
    System.out.println("logged keyspace: “+ session.getLoggedKeyspace());
     // change keyspace to cms
     String changeKeySpaceQuery = "USE "+keyspaceName;
     // execute command
     session.execute(changeKeySpaceQuery);
     // print current keyspace
     System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
    String query ="CREATE COLUMNFAMILY "+ keyspaceName +"."+columnFamily+"("+
                "listingId varchar,"+
                "sellerId Varchar,"+
                "skuId varchar,"+
                "productId varchar,"+
                "mrp int,"+
                "ssp int,"+
                "sla int,"+
                "stock int,"+
                "title text,"+
                "PRIMARY KEY (productId,listingId));

    // execute the query
    session.execute(query);
}
```

get the session object

the method returns the current keyspace for the session

it is currently null

it is currently null

```
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();
     System.out.println("logged keyspace: “+ session.getLoggedKeyspace());
     // change keyspace to cms
    String changeKeySpaceQuery = "USE "+keyspaceName;
    // execute command
    session.execute(changeKeySpaceQuery);
    // print current keyspace
    System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
    String query = "CREATE COLUMNFAMILY "+columnFamily+ "("+
                    "listingId varchar,"+
                    "sellerId Varchar,"+
                    "skuId varchar,"+
                    "productId varchar,"+
                    "mrp int,"+
                    "ssp int,"+
                    "sta int,"+
                    "stock int,"+
                    [illegible]
                    "PRIMARY KEY (productId, listingId));";

    // execute the query
    session.execute(query);
}
```

the state of the session is similar to

```
cassandra@cqlsh>
```

to set the value of logged keyspace we will execute the use keyspace command

changing the logged keyspace to cms

```
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();
    System.out.println(session.getLoggedKeyspace());

    // change keyspace to cms
    String changeKeySpaceQuery = "USE "+keyspaceName;
    // execute command
    session.execute(changeKeySpaceQuery);
    // print current keyspace
    System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
    String query = "CREATE COLUMNFAMILY "+ keyspaceName +"."+columnFamily+ "("+
                    "listingId varchar,"+
                    "sellerId Varchar,"+
                    "skuId varchar,"+
                    "productId varchar,"+
                    "mrp int,"+
                    "ssp int,"+
                    "sla int,"+
                    "stock int,"+
                    "title text,"+
                    "PRIMARY KEY (productId, listingId));";

    // execute the query
```

we pass keyspaceName = cms to the method

```
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();
    System.out.println(session.getLoggedKeyspace());

    // change keyspace to cms
      String changeKeySpaceQuery = "USE "+keyspaceName;
      // execute command
      session.execute(changeKeySpaceQuery);
      // print current keyspace
      System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
    String query = "CREATE COLUMNFAMILY "+keyspaceName+"."+columnFamily+ "("+
                    "listingId varchar,"+
                    "sellerId Varchar,"+
                    "skuId varchar,"+
                    "productId varchar,"+
                    "mrp int,"+
                    "ssp int,"+
                    "sla int,"+
                    "stock int,"+
                    "title text,"+
                    "PRIMARY KEY (productId, listingId));";

    // execute the query
```

the output is

```
logged keyspace :cms
```

current keyspace is now cms

```
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();
    System.out.println(session.getLoggedKeyspace());

    // change keyspace to cms
      String changeKeySpaceQuery = "USE "+keyspaceName;
      // execute command
      session.execute(changeKeySpaceQuery);
      // print current keyspace
      System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
    String query = "CREATE COLUMNFAMILY "+ keyspaceName +"."+columnFamily+ "("+
                    "listingId varchar,"+
                    [illegible]
                    "skuId varchar,"+
                    "productId varchar,"+
                    [illegible]
                    "ssp int,"+
                    "sla int,"+
                    [illegible]
                    "title text,"+
                    "PRIMARY KEY (productId, listingId));";

    // execute the query
```

In queries where you do not pass the keyspaceName as parameter, its good to check the current keyspace value in the session

execute the query to create columnfamily

```
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();
    System.out.println(session.getLoggedKeyspace());

    // change keyspace to cms
    String changeKeySpaceQuery = "USE "+keyspaceName+";";
    // execute command
    session.execute(changeKeySpaceQuery);
    // print current keyspace
    System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
  String query = "CREATE COLUMNFAMILY "+ keyspaceName +"."+columnFamily+ "("+
                    "listingId varchar,"+
                    "sellerId Varchar,"+
                    "skuId varchar,"+
                    "productId varchar,"+
                    "mrp int,"+
                    "ssp int,"+
                    "sla int,"+
                    "stock int,"+
                    "title text,"+
                    "PRIMARY KEY (productId, listingId));";
       // execute the query
    session.execute(query);
}
```

Java

```java
static void createColumnFamily(String keyspaceName, String columnFamily){
    // building the query to create columnfamily
    Session session = Connector.getSession();

    // change keyspace to cms
    String changeKeySpaceQuery = "USE "+keyspaceName;
    // execute command
    session.execute(changeKeySpaceQuery);
    // print current keyspace
    System.out.println(session.getLoggedKeyspace());

    // query to create columnfamily
    String query = "CREATE COLUMNFAMILY "+ keyspaceName +"."+columnFamily+ "("+
                    "listingId varchar,"+
                    "sellerId Varchar,"+
                    "skuId varchar,"+
                    "productId varchar,"+
                    "mrp int,"+
                    "ssp int,"+
                    "sla int,"+
                    "stock int,"+
                    "title text,"+
                    "PRIMARY KEY (productId, listingId));";

    // execute the query
    session.execute(query);
}
```

Let's check whether the column family has been created

Code to check if Column Family exists

```
public class Main {

  static void checkIfColumnFamilyCreated(String keyspace, String cfName){
      //get session
      Session session = Connector.getSession();
      // get cluster
      Cluster cluster = session.getCluster();
      // get all keyspaces in cluster
      List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
      if(keyspaceMetadatas != null ){
          // iterate over keyspaces
          for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
              if(keyspace.equals(keyspaceMetadata.getName())) {
                  if (keyspaceMetadata.getTable(cfName) != null) {
                      System.out.println("Column Family :"+cfName + " exists in keyspace :"+keyspace);
                  }
              }

          }
      }
      System.out.println("Column Family :"+cfName + " doesnt exist");;
  }
}
```

To check if a CF exists in a keyspace

1. Get session object
2. Get the cluster object
3. Get all keyspace data
4. Check if CF exists in keyspace

```
public class Main{
   static void checkIfColumnFamilyExists(String keyspace, String cfName){
      //get session
      Session session = Connector.getSession();
      // get cluster
      Cluster cluster = session.getCluster();
      // get all keyspaces in cluster
      List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
      if(keyspaceMetadatas != null ){
         // iterate over keyspaces
         for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
            if(keyspace.equals(keyspaceMetadata.getName()))
               if (keyspaceMetadata.getTable(cfName) != null) {
                  System.out.println("Column Family :"+cfName + " exists in
   keyspace :"+keyspace);
            }

         }
      }
      System.out.println("Column Family :"+cfName + " doesnt exist");;
   }
}
```

Let's go through the code now

1.Get session object

```
public class Main {
 static void checkIfColumnFamilyCreated(String keyspace, String cfName){
    //get session
     Session session = Connector.getSession();
     // get cluster

     Cluster cluster = session.getCluster();
    // get all keyspaces in cluster
    List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
    if(keyspaceMetadatas != null) {
        // iterate over keyspaces
        for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
            if(keyspace.equals(keyspaceMetadata.getName())) {
                if (keyspaceMetadata.getTable(cfName) != null){
                    System.out.println("Column Family :"+cfName + " exists in
 keyspace :"+keyspace);
                }
            }

        }
    }
    System.out.println("Column Family :"+cfName + " doesnt exist");
}
}
```

As before, we get the session object from the connector module

2.Get the cluster object

```
public class Main {
static void checkIfColumnFamilyCreated(String keyspace, String cfName){
     //get session
    Session session = Connector.getSession();
      // get cluster
      Cluster cluster = session.getCluster();
    // get all keyspaces in cluster
    List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
    if(keyspaceMetadatas != null ){
        // iterate over keyspaces
        for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
            if(keyspace.equals(keyspaceMetadata.getName())){
                if (keyspaceMetadata.getTable(cfName) != null) {
                    System.out.println("Column Family :"+cfName + " exists in
keyspace :"+keyspace);
                }
            }

        }
    }
    System.out.println("Column Family :"+cfName + " doesnt exist");
  }
}
```

session object contains the cluster to which it is connected

2.Get the cluster object

```
public class Main {
 static void checkIfColumnFamilyCreated(String keyspace, String cfName){
     //get session
    Session session = Connector.getSes[illegible]
      // get cluster
     Cluster cluster = session.getCluster();
    // get all keyspaces in cluster
    List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
    if(keyspaceMetadatas != null ){
      [illegible]
        for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
          [illegible]etadata.getName())) {
            if (keyspaceMetadata.getTable(cfName) != nu[illegible]
                System.out.println("Column Family :"+cfName + " exists in
keyspace :"+keyspace);
              [illegible]
          }
        }
    }
    S[illegible]me + " doesnt exis[illegible]");
  }
}
```

cluster contains

information about the nodes in the cluster

token ranges for partition

retry policies to handle read/write timeouts

connection pooling options

keyspace configurations

and other configurations of cluster

2.Get the cluster object

```
public class Main {
  static void checkIfColumnFamilyCreated(String keyspace, String cfName){
    //get session
      Session session = Connector.getSe[illegible]
      // get cluster
      Cluster cluster = session.getCluster();
    // get all keyspaces in cluster
    List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
    if(keyspaceMetadatas != null ){
      // iterate over keyspaces
      for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
        [illegible]
          [illegible] null) {
            System.out.println("Column Family :"+cfN[illegible]
keyspac[illegible]"+ke[illegible]
          }
        }
      }
    System.out.println("Column Family :"+cfName + " doesnt exist");
  }
}
```

cluster contains

all this data is clubbed in different classes

cluster object contains the methods to access them

3.Get all keyspace data

```
public class Main {
  static void checkIfColumnFamilyCreated(String keyspace, String cfName){
    //get session
    Session session = Connector.getSession();
    // get cluster
    Cluster cluster = session.getCluster();
    // get all keyspaces in cluster
      List<KeyspaceMetadata> keyspaceMetadatas =
cluster.getMetadata().getKeyspaces();
    if(keyspaceMetadatas != null ){
        // iterate over keyspaces
        for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
            if(keyspace.equals(keyspaceMetadata.getName()) ) {
                if (keyspaceMetadata.getTable(cfName)!= null) {
                    System.out.println("Column Family:"+cfName+" exists in keyspace:"+keyspace);
                }
            }

        }
    }
    System.out.println("Column Family "+cfName+" doesn't exist");
  }
}
```

data about keyspace is in the metadata class

4.Check if CF exists in keyspace

```
public class Main {
  static void checkIfColumnFamilyCreated(String keyspace, String cfName){
    //get session
    Session session = Connector.getSession();
    // get cluster
    Cluster cluster = session.getCluster();
    // get all keyspaces in cluster
    List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
    if(keyspaceMetadatas != null ){
          // iterate over keyspaces
          for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
                if(keyspace.equals(keyspaceMetadata.getName())) {
                if (keyspaceMetadata.getTable(cfName) != null) {
                    System.out.println("Column Family :"+cfName + " exists in
keyspace :"+keyspace);
                }
          }
    }
    System.out.println("Column Family :"+cfName + " [illegible] exist");
  }
}
```

we iterate over all keyspaces until we reach the required keyspace

4.Check if CF exists in keyspace

```
public class Main {
  static void checkIfColumnFamilyCreated(String keyspace, String cfName){
    //get session
    Session session = Connector.getSession();
    // get cluster
    Cluster cluster = session.getCluster();
    // get all keyspaces in cluster
    List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
    if(keyspaceMetadatas != null ){
        // iterate over keyspaces
        for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
            if(keyspace.equals(keyspaceMetadata.getName())) {
                if (keyspaceMetadata.getTable(cfName) != null) {
                    System.out.println("Column Family :"+cfName + " exists in
keyspace :"+keyspace);
                }
            }
        }
    }
    System.out.println("Column Family :"+cfName + " doesn't exist
  }
}
```

once we reach the required keyspace, we check if it contains our column family

```
public class Main {
  static void checkIfColumnFamilyCreated(String keyspace, String cfName){
    //get session
    Session session = Connector.getSession();
    // get cluster
    Cluster cluster = session.getCluster();
    // get all keyspaces in cluster
    List<KeyspaceMetadata> keyspaceMetadatas = cluster.getMetadata().getKeyspaces();
    if(keyspaceMetadatas != null ){
        // iterate over keyspaces
        for(KeyspaceMetadata keyspaceMetadata : keyspaceMetadatas){
            if(keyspace.equals(keyspaceMetadata.getName())) {
                if (keyspaceMetadata.getTable(cfName) != null) {
                    System.out.println("Column Family :"+cfName + " exists in
keyspace :"+keyspace);
                }
            }

        }
    }
    System.out.println("Column Family :"+cfName + " doesnt exist");;
  }
}
```

Let's add furniture pricing data to listings

Let's first understand the code flow

We have an enum class which contains possible attributes for listing and product

```
public enum AttributeNames {

    SELLERID("sellerid"),
    PRODUCTID("productid"),
    LISTINGID("listingid"),
    SKUID("skuid"),
    MRP("mrp"),
    SSP("ssp"),
    SLA("sla"),
    TITLE("title"),
    BRAND("brand"),
    MODELID("modelid"),
    KEYFEATURES("keyfeatures"),
    PINCODESSERVED("pincodes_served"),
    LENGTH("length"),
    BREADTH("breadth"),
    HEIGHT("height"),
    PUBLISHER("publisher"),
    CATEGORY("category"),
    STOCK("stock");
```

We will refer to this class when we want to use attribute names

Let's have a look at the listing class

```java
public class Listing {
    String listingId;
    Map<String, Object> attributes = Maps.newHashMap();


    public Listing() {
    }

    public String getListingId() {
        return listingId;
    }

    public void setListingId(String listingId) {
        this.listingId = listingId;
    }


    public Map<String, Object> getAttributes() {
        return attributes;
    }

    public void setAttributes(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String toString() {
        return "Listing{" +
                "listingId='" + listingId + '\'' +
                ", attributes=" + attributes +
                '}';
    }
}
```

This is a listing class which represents one listing

Let's have a look at the listing class

```java
public class Listing {
    String listingId;
    Map<String, Object> attributes = Maps.newHashMap();

    public Listing() {
    }

    public String getListingId() {
        return listingId;
    }

    public void setListingId(String listingId) {
        this.listingId = listingId;
    }


    public Map<String, Object> getAttributes() {
        return attributes;
    }

    public void setAttributes(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String toString() {
        return "Listing{" +
                "listingId='" + listingId + '\'' +
                ", attributes=" + attributes +
                '}';
    }
}
```

Listing attributes are stored in a map

Let's have a look at the listing class

```java
public class Listing {
    String listingId;
        Map<String, Object> attributes = Maps.newHashMap();

    public Listing() {
    }

    public String getListingId() {
        return listingId;
    }

    public void setListingId(String listingId) {
        this.listingId = listingId;
    }


    public Map<String, Object> getAttributes() {
        return attributes;
    }

    public void setAttributes(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String toString() {
        return "Listing{" +
                "listingId='" + listingId + '\'' +
                ", attributes=" + attributes +
                '}';
    }
}
```

attribute name -> key
attribute data ->value

attributes:
{
  ‘stock’:5
  ‘sla’:2,
  ‘sellerid’:’Fab’
}

Let's have a look at the listing class

```java
public class Listing {
    String listingId;
    Map<String, Object> attributes = Maps.newHashMap();


    public Listing() {
    }

    public String getListingId() {
        return listingId;
    }

    public void setListingId(String listingId) {
        this.listingId = listingId;
    }


    public Map<String, Object> getAttributes() {
        return attributes;
    }

    public void setAttributes(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String toString() {
        return "Listing{" +
                "listingId='" + listingId + '\'' +
                ", attributes=" + attributes +
                '}';
    }
}
```

The listing id is not part of the attribute map

It is a way to uniquely identify a listing

Let's have a look at the listing class

```java
public class Listing {
    String listingId;
    Map<String, Object> attributes = Maps.newHashMap();


    public Listing() {
    }

    public String getListingId() {
        return listingId;
    }

    public void setListingId(String listingId) {
        this.listingId = listingId;
    }


    public Map<String, Object> getAttributes() {
        return attributes;
    }

    public void setAttributes(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    @Override
    public String toString() {
        return "Listing{" +
                "listingId='" + listingId + '\'' +
                ", attributes=" + attributes +
                '}';
    }
}
```

The rest of the methods are simple getters and setters

PersistenceHandler class has the methods to perform database operations

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = "listings";

   public void put(Listing listing){
      Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

     Map<String, Object> attributes = listing.getAttributes();
     insertStatement.value(AttributeNames.LISTINGID.getValue(), listing.getListingId());
     for(String attributeName : attributes.keySet()){
         insertStatement = insertStatement.value(attributeName,
 attributes.get(attributeName));
     }
     insertStatement.setDefaultTimestamp(new
 ThreadLocalMonotonicTimestampGenerator().next());
     Session session = Connector.getSession();
     session.execute(insertStatement);
 }
 }
```

We trigger the operation from the Main class

```java
public class Main {
    public static void main(String[] args) {
       insertDataToListing();
    }
    static void insertDataToListing(){
    // populating attributes
    Listing listing = createListingData();
    ListingPersistenceHandler listingPersistenceHandler = new
ListingPersistenceHandler();
    // put data to listings
    listingPersistenceHandler.put(listing);
  }

static Listing createListingData(){
    Listing listing = new Listing();
    Map<String, Object> attributes = Maps.newHashMap();
    listing.setListingId("LISTINGFABSOFA5");
    attributes.put(AttributeNames.SELLERID.getValue(), "Fab");
    attributes.put(AttributeNames.SKUID.getValue(), "SKU2");
    attributes.put(AttributeNames.MRP.getValue(), 5000);
    attributes.put(AttributeNames.SSP.getValue(), 4000);
    attributes.put(AttributeNames.SLA.getValue(), 2);
    attributes.put(AttributeNames.STOCK.getValue(), 2);
    attributes.put(AttributeNames.PRODUCTID.getValue(), "SOFA5");
    attributes.put(AttributeNames.TITLE.getValue(), "Urban Loving Sofa 3 Seater");
    listing.setAttributes(attributes);
    return listing;

}
}
```

Main creates and inserts listing data

We trigger the operation from the Main class

Main class creates the listing data

```
public class Main {
    public static void main(String[] args) {
       insertDataToListing();
    }
    static void insertDataToListing(){
    // populating attributes
        Listing listing = createListingData();
      ListingPersistenceHandler listingPersistenceHandler = new
ListingPersistenceHandler();
    // put data to listings
    listingPersistenceHandler.put(listing);
  }

static Listing createListingData(){
    Listing li[illegible]ng = [illegible] Listing();
    Map<String[illegible]ibutes[illegible]
    listing.se[illegible]"[illegible]STI[illegible]");
    attributes.put(AttributeNames.SELLERID.getVal[illegible]), "Fab");
    attributes.put(AttributeNames.SKUID.getValue(), "SKU2[illegible]);
    attributes.put(Attrib[illegible]es.MRP.getValue[illegible]000[illegible]
    attributes.put(Attrib[illegible]es.S[illegible]), 000[illegible]
    attributes.put(Attrib[illegible]es.S[illegible]), [illegible]);
    attributes.put(AttributeNames.STOCK.getValue(), 2);
    attributes.put(Attribute[illegible]es.PR[illegible]TID.getValue(), "SOF[illegible]5");
    attributes.put(Attribute[illegible]
    listing.setAttributes(at[illegible]es);
    return listing;

}
}
```

Calls ListingPersistenceHandler to insert the data to the listings column family

Let's see what we do to insert the data

1. Create data
2. Set up prepared statement
3. Execute the statement

1. Create data

```java
public class Main {
    public static void main(String[] args){
       insertDataToListing();
    }

   static Listing createListingData(){
    Listing listing = new Listing();
    Map<String, Object> attributes = Maps.newHashMap();
    listing.setListingId("LISTINGFABSOFA5");
    attributes.put(AttributeNames.SELLERID.getValue(), "Fab");
    attributes.put(AttributeNames.SKUID.getValue(), "SKU2");
    attributes.put(AttributeNames.MRP.getValue(), 5000);
    attributes.put(AttributeNames.SSP.getValue(), 4000);
    attributes.put(AttributeNames.SLA.getValue(), 2);
    attributes.put(AttributeNames.STOCK.getValue(), 2);
    attributes.put(AttributeNames.PRODUCTID.getValue(), "SOFA5");
    attributes.put(AttributeNames.TITLE.getValue(), "Urban Loving Sofa 3 Seater");
    listing.setAttributes(attributes);
    return listing;

}
  static void insertDataToListing(){
    // populating attributes
    Listing listing = createListingData();
    ListingPersistenceHandler listingPersistenceHandler = new ListingPersistenceHandler();
    // put data to listings
    listingPersistenceHandler.put(listing);
  }


}
```

Create the listing

Insert it into the database

1. Create data

```
public class Main {
    public static void main(String[] args){
        insertDataToListing();
    }
```

```
    static Listing createListingData(){
     Listing listing = new Listing();
     Map<String, Object> attributes = Maps.newHashMap();
     listing.setListingId("LISTINGFABSOFA5");
     attributes.put(AttributeNames.SELLERID.getValue(), "Fab");
     attributes.put(AttributeNames.SKUID.getValue(), "SKU2");
     attributes.put(AttributeNames.MRP.getValue(), 5000);
     attributes.put(AttributeNames.SSP.getValue(), 4000);
     attributes.put(AttributeNames.SLA.getValue(), 2);
     attributes.put(AttributeNames.STOCK.getValue(), 2);
     attributes.put(AttributeNames.PRODUCTID.getValue(), "SOFA5");
     attributes.put(AttributeNames.TITLE.getValue(), "Urban Loving
Sofa 3 Seater");
     listing.setAttributes(attributes);
     return listing;

}
```

Set up a new listing object

Add a bunch of attributes to it

Return the listing

1. Create data

```
public class Main {
    public static void main(String[] args){
       insert[illegible]aToListing();
    }

   static Lis[illegible]g[illegible]re[illegible]ng[illegible]ta[illegible]{
    Listing listing = new Listing();
    Map<String, Object> attributes = Maps.newHashMap();
    listing.setListingI[illegible]"LISTI[illegible]FABSOFA5");
    at[illegible]put([illegible]Names.[illegible].[illegible]e(), "Fab");
    at[illegible]put(A[illegible]Nam[illegible]s.[illegible]UI[illegible]value(), "SKU2");
    attributes.put(AttributeNames.MRP.getValue(), 5000);
    attributes.put(AttributeNames.SSP.getValue(), 4000);
    attributes.put(AttributeNam[illegible]LA.getValue(), 2);
    attributes[illegible]T[illegible]value(), 2);
    attributes[illegible]D.getValue(), "SOFA5");
    attributes[illegible]tValue(), "Urban Loving Sofa 3 Seater");
    listing.setAttributes(attributes);
    return listing;

}
```

Create a persistence handler

Insert the data into the database

```
  static void insertDataToListing(){
    // populating attributes
    Listing listing = createListingData();
    ListingPersistenceHandler listingPersistenceHandler = new
ListingPersistenceHandler();
    // put data to listings
    listingPersistenceHandler.put(listing);
  }
```

Let's see what we do to insert the data

1. Create data
2. Set up prepared statement
3. Execute the statement

2. Set up prepared statement

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = "listings";

   public void put(Listing listing){
      Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

     Map<String, Object> attributes = listing.getAttributes();
     insertStatement.value(AttributeNames.LISTINGID.getValue(), listing.getListingId());
     for(String attributeName : attributes.keySet()){
         insertStatement = insertStatement.value(attributeName,
 attributes.get(attributeName));
     }
     insertStatement.setDefaultTimestamp(new
 ThreadLocalMonotonicTimestampGenerator().next());
     Session session = Connector.getSession();
     session.execute(insertStatement);
 }
 }
```

2. Set up prepared statement

we have defined keyspace
and CF for listings here

```
public class ListingPersistenceHandler {

  private static String keyspace = "cms";
  private static String columnFamily = "listings";

   public void put(Listing listing){
       Insert insertStatement =
 QueryBuilder.insertInto(keyspace, columnFamily);

       Map<String, Object> attributes = listing.getAttributes();
       insertStatement.value(AttributeNames.LISTINGID.getValue(), listing.getListingId());
       for(String attributeName : attributes.keySet()){
           insertStatement = insertStatement.value(attributeName, attributes.get(attributeName));
       }
       insertStatement.setDefaultTimestamp(System.currentTimeMillis());
       Session session = Connector.getSession();
       session.execute(insertStatement);
   }
}
```

QueryBuilder class provides
the methods to build queries

2. Set up prepared statement

Insert is an executable query
with all query options

```
public class ListingPersistenceHandler {

  private static String keyspace = "cms";
  private static String columnFamily = "listings";

    public void put(Listing listing){
        Insert insertStatement =
QueryBuilder.insertInto(keyspace, columnFamily);

    Map<String, Object> attributes = listing.getAttributes();
    insertStatement.value(AttributeNames.LISTINGID.getValue(), listing.getListingId());
    for(String attributeName : attributes.keySet()){
        insertStatement = insertStatement.value(attributeName, attributes.get(attributeName));
    }
    insertStatement.setDefaultTimestamp(System.currentTimeMillis());
    Session session = Connector.getSession();
    session.execute(insertStatement);
  }
}
```

the options in the insert
statement - e.g. using timestamp
are the methods in Insert class

2. Set up prepared statement

parameters to be inserted

```
public class ListingPersistenceHandler {

   private static String keyspace = "cms";
   private static String columnFamily = "listings";

    public void put(Listing listing){
     Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

    Map<String, Object> attributes = listing.getAttributes();

  insertStatement.value(AttributeNames.LISTINGID.getValue(),
  listing.getListingId());

  for(String attributeName : attributes.keySet()){
          insertStatement =
  insertStatement.value(attributeName,
  attributes.get(attributeName));
      }
    insertStatement.setDefaultTimestamp(System.currentTimeMillis());
    Session session = Connector.getSession();
    session.execute(insertStatement);
  }
  }
```

attribute name

2. Set up prepared statement

parameters to be inserted

```
public class ListingPersistenceHandler {

   private static String keyspace = "cms";
   private static String columnFamily = "listings";

    public void put(Listing listing){
    Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

    Map<String, Object> attributes = listing.getAttributes();

  insertStatement.value(AttributeNames.LISTINGID.getValue(),
  listing.getListingId());

  for(String attributeName : attributes.keySet()){
          insertStatement =
  insertStatement.value(attributeName,
  attributes.get(attributeName));
      }
      insertStatement.setDefaultTimestamp(System.currentTimeMillis());
      Session session = Connector.getSession();
      session.execute(insertStatement);
  }
  }
```

attribute value

2. Set up prepared statement

parameters to be inserted

```
public class ListingPersistenceHandler {

   private static String keyspace = "cms";
   private static String columnFamily = "listings";

    public void put(Listing listing){
    Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

    Map<String, Object> attributes =
  listing.getAttributes();

  insertStatement.value(AttributeNames.LISTINGID.getVa
  lue(), listing.getListingId());

  for(String attributeName : attributes.keySet()){
          insertStatement =
  insertStatement.value(attributeName,
  attributes.get(attributeName));
      }
      insertStatement.setDefaultTimestamp(System.currentTimeMillis());
      Session session = Connector.getSession();
      session.execute(insertStatement);
  }
  }
```

We iterate over attributes and add the attribute name/value pairs to the insertStatement

2. Set up prepared statement

```
public class ListingPersistenceHandler {

  private static String keyspace = "cms";
  private static String columnFamily =
"listings";

   public void put(Listing listing){
   Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

    Map<String, Object> attributes = listing.getAttributes();
    insertStatement.value(AttributeNames.LISTINGID.getValue(), listing.getListingId());

 for(String attributeName : attributes.keySet()){
        insertStatement = insertStatement.value(attributeName, attributes.get(attributeName));
    }
   insertStatement.setDefaultTimestamp(new
 ThreadLocalMonotonicTimestampGenerator().next()
 );
    Session session = Connector.getSession();
    session.execute(insertStatement);
 }
 }
```

to send the timestamp with the query

the option is same as USING TIMESTAMP in cql

2. Set up prepared statement

```
public class ListingPersistenceHandler {

  private static String keyspace = "cms";
  private static String columnFamily =
"listings";

   public void put(Listing listing){
    Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

     Map<String, Object> attributes = listing.getAttributes();
     insertStatement.value(AttributeNames.LISTINGID.getValue(), listing.getListingId());

 for(String attributeName : attributes.keySet()){
         insertStatement = insertStatement.value(attributeName,
 attributes.get(attributeName));
     }
     insertStatement.setDefaultTimestamp(new
 ThreadLocalMonotonicTimestampGenerator().next
 ()
 );
     Session session = Connector.getSession();
     session.execute(insertStatement);
 }
 }
```

timestamp should be in microseconds

cassandra-driver has timestampGenerators classes which do that

Let's see what we do to insert the data

1. Create data
2. Set up prepared statement
3. Execute the statement

3. Execute the statement

```
public class ListingPersistenceHandler {

   private static String keyspace = "cms";
   private static String columnFamily =
"listings";

    public void put(Listing listing){

    Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

     Map<String, Object> attributes = listing.getAttributes();
     insertStatement.value(AttributeNames.LISTINGID.getValue(),
 listing.getListingId());

 for(String attributeName : attributes.keySet()){
         insertStatement = insertStatement.value(attributeName,
 attributes.get(attributeName));
     }
     insertStatement.setDefaultTimestamp(new
 ThreadLocalMonotonicTimestampGenerator().next()
 );
     Session session = Connector.getSession();
        session.execute(insertStatement);
 }
 }
```

once the statement is prepared

we get the session object and execute the statement

3. Execute the statement

```
public class ListingPersistenceHandler {

  private static String keyspace = "cms";
  private static String columnFamily =
"listings";

   public void put(Listing listing){

   Insert insertStatement = QueryBuilder.insertInto(keyspace, columnFamily);

    Map<String, Object> attributes = listing.getAttributes();
    insertStatement.value(AttributeNames.LISTINGID.getValue(),
 listing.getListingId());

 for(String attributeName : attributes.keySet()){
        insertStatement = insertStatement.value(attributeName,
 attributes.get(attributeName));
    }
    insertStatement.setDefaultTimestamp(new
 ThreadLocalMonotonicTimestampGenerator().next()
 );
   Session session = Connector.getSession();
     session.execute(insertStatement);
 }
 }
```

Creating queries using string concatenation is not a good idea - we did that earlier

Its a good practice to use the QueryBuilder to prepare queries

Let's see what we do to insert the data

1. Create data
2. Set up prepared statement
3. Execute the statement

Let's search for Products by different categories and brands

Main.java Class

Let's create column family "products" for cms

```java
public class Main {

public static void main(String[] args){
{
   createProductColumnFamily("cms");

}
static void createProductColumnFamily(String keyspaceName){

    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("+
            "productId varchar,"+
            "brand varchar,"+
            "length int,"+
            "breadth int,"+
            "height int,"+
            "category varchar,"+
            "title text,"+
            "publisher text,"+
            "keyfeatures list<text>,"+
            "PRIMARY KEY (category, brand, productId));";

    session.execute(query);

  }
}
```

We will create the column family like the way we did with listings

In the Main class, we create a method to create the products CF

Note the schema design we are using for products this time

Main.java Class

Let's create column family "products" for cms

Our primary key has 3 columns

```
public class Main {

public static void main(String[] args){
{
   createProductColumnFamily("cms");

}
static void createProductColumnFamily(String keyspaceName){

    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("+
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category, brand,
productId));";
```

We've determined this based on the kind of queries we want to make on this column family

Main.java Class

Let's create column family "products" for cms

```
public class Main {

public static void main(String[] args){
{
   createProductColumnFamily("cms");

}
static void createProductColumnFamily(String keyspaceName){

    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY "+ columnFamily+ "("+
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category,
brand,  productId));";
```

We have added a new column - category

Category is our partition key

Main.java Class

Let's create column family "products" for cms

We have added a new column - category

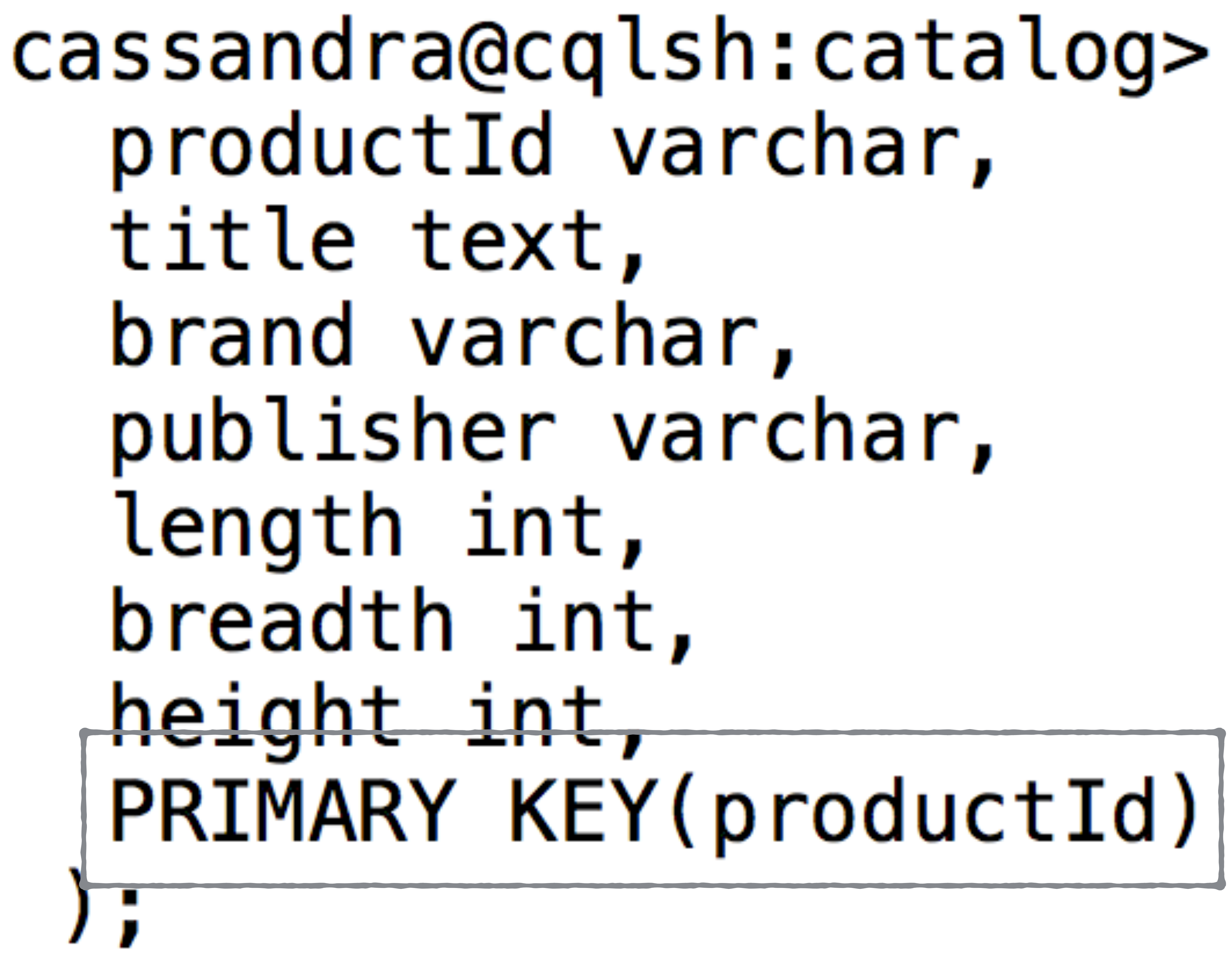
Category is our partition key

brand and productid our clustering columns

```
public class Main {

public static void main(String[] args)
{
    createProductColumnFamily("cms");
}
static void createProductColumnFamily(String keyspaceName){

    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("+
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category,
brand,  productId));";
```

RECAP

product definition of catalog keyspace product CF



```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
 );
```

Due to cassandra's restrictions on primary keys, we were able to perform operations only on productid

e.g. CRUD with a list of productIds

product definition of catalog keyspace

```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
 );
```

```
public class Main {
public static void main(String[] args){
{
createProductColumnFamily("cms");
}
static void createProductColumnFamily(String keyspaceName){

    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category, brand,
productId));";
```

But with this new primary key definition, we can perform queries using category and brand

product definition of catalog keyspace

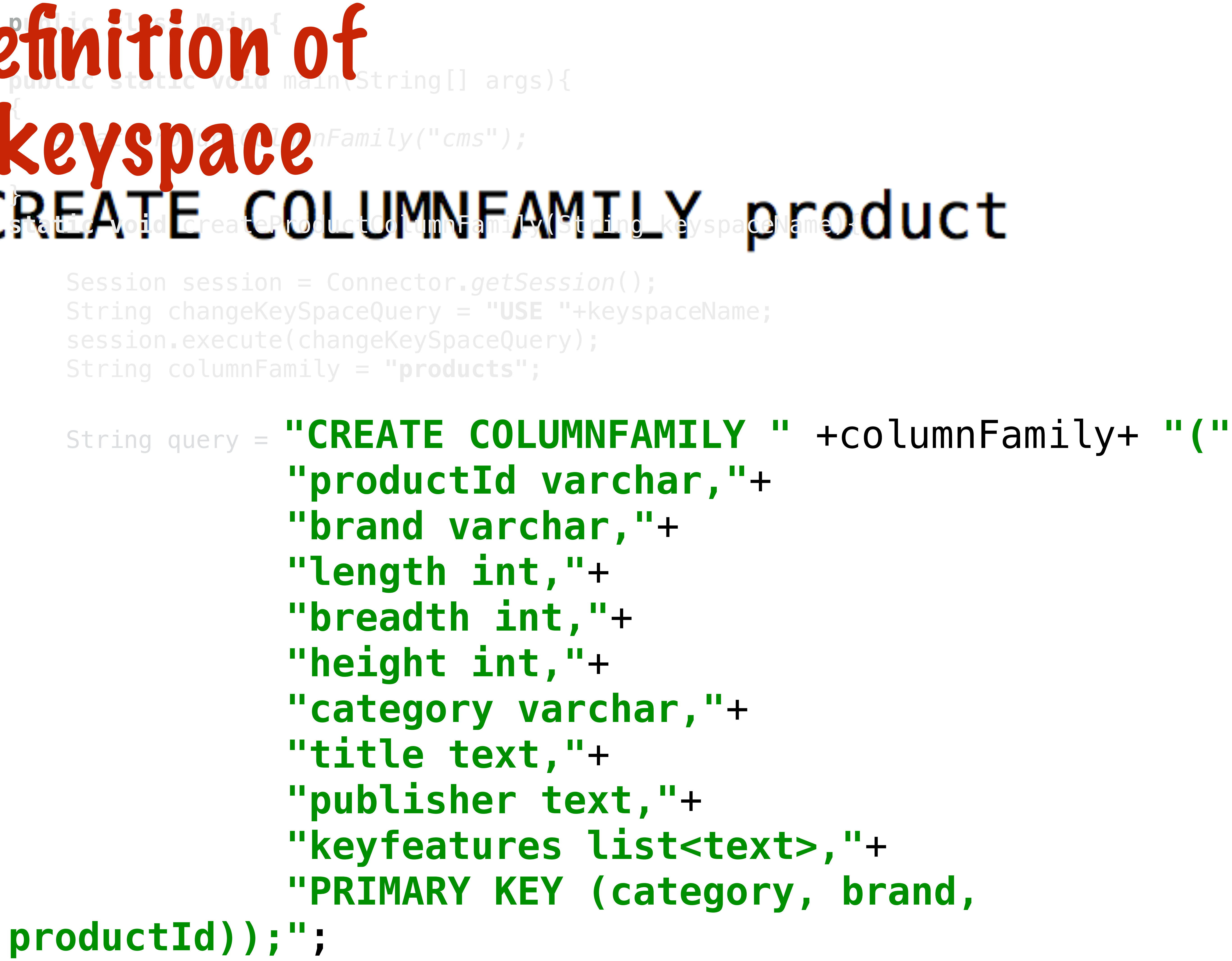
```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
 );
```

```
public class Main {
public static void main(String[] args){
{
createProductColumnFamily("cms");
}
static void createProductColumnFamily(String keyspaceName){

   Session session = Connector.getSession();
   String changeKeySpaceQuery = "USE "+keyspaceName;
   session.execute(changeKeySpaceQuery);
   String columnFamily = "products";

   String query = "CREATE COLUMNFAMILY " +columnFamily+ "("
                  "productId varchar,"+
                  "brand varchar,"+
                  "length int,"+
                  "breadth int,"+
                  "height int,"+
                  "category varchar,"+
                  "title text,"+
                  "publisher text,"+
                  "keyfeatures list<text>,"+
                  "PRIMARY KEY (category, brand,
productId));";
```

e.g. bulk update on products of the same category and brand

product definition of catalog keyspace

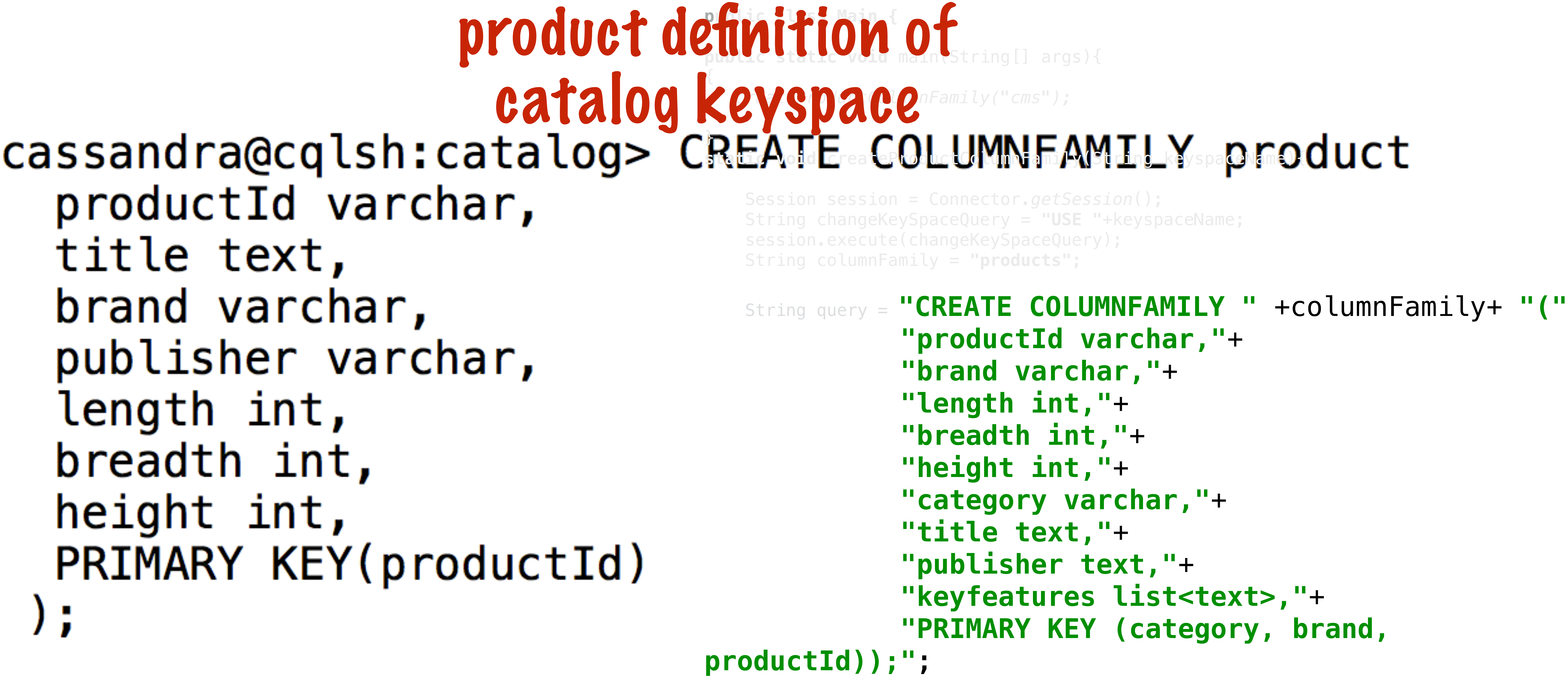
```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
 );
```



```
public class Main {
public static void main(String[] args){
{
createProductColumnFamily("cms");
}
static void createProductColumnFamily(String keyspaceName){

    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category, brand,
productId));";
```

or listing products of a given category and brand

product definition of catalog keyspace



```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
 );
```

```
public class Main {
public static void main(String[] args){
{
    createProductColumnFamily("cms");
}
static void createProductColumnFamily(String keyspaceName){
    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category, brand,
productId));";
```

you can choose to define your column family in either way

product definition of catalog keyspace

```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
);
```

```
public class Main {
public static void main(String[] args){
{
    createProductColumnFamily("cms");
}
static void createProductColumnFamily(String keyspaceName){
    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category, brand,
productId));";
```

In cassandra, we model our CF definition based on our QUERIES and DATA DISTRIBUTION

product definition of catalog keyspace

```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
 );
```

```
public class Main {
public static void main(String[] args){
{
        createP...nFamily("cms");
}
static void createProductColumnFamily(String keyspaceName){

    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category, brand,
productId));";
```

This model works if we only perform CRUD operations

this model works if we search data based on category and brand

product definition of catalog keyspace

```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
);
```

```
public class Main {
public static void main(String[] args){
{
createProductColumnFamily("cms");
}
static void createProductColumnFamily(String keyspaceName){

    Session session = Connector.getSession();
    String changeKeySpaceQuery = "USE "+keyspaceName;
    session.execute(changeKeySpaceQuery);
    String columnFamily = "products";

    String query = "CREATE COLUMNFAMILY " +columnFamily+ "("
                   "productId varchar,"+
                   "brand varchar,"+
                   "length int,"+
                   "breadth int,"+
                   "height int,"+
                   "category varchar,"+
                   "title text,"+
                   "publisher text,"+
                   "keyfeatures list<text>,"+
                   "PRIMARY KEY (category, brand,
productId));";
```

This can be used only if the categories are uniformly distributed

because as the partition key it is responsible for distributing data across nodes in the cluster

Coming back to our example

We have created the products CF

Let's add some products

Let's add some products

We add products in db in 2 steps

1. create a product data object

2. ProductPersistenceHandler inserts the product in db

We add products in db in 2 steps

1. create a product data object
2. ProductPersistenceHandler inserts the product in db

AddProductsMain.java

1. create a product data object

For product SOFA1

set attribute data in attributeMaps1

key of attributesMaps1 is columnName

value of attributesMaps1 is data for the column

```
public static void main(String[] args){
    addData();
}

static void addData(){
    ProductPersistenceHandler persistenceHandler = new
ProductPersistenceHandler();
    Product product1 = new Product();
    Map<String, Object> attributesMaps1 = Maps.newHashMap();
    attributesMaps1.put(CATEGORY.getValue(), "sofa");
    attributesMaps1.put(BRAND.getValue(), "Fab");
    attributesMaps1.put(BREADTH.getValue(), 100);
    attributesMaps1.put(HEIGHT.getValue(), 200);
    attributesMaps1.put(LENGTH.getValue(), 500);
    attributesMaps1.put(TITLE.getValue(), "Urban Living Derby");
    product1.setProductId("SOFA1");
    product1.setAttributesMap(attributesMaps1);
    persistenceHandler.insertProducts(product1);
}
```

AddProductsMain.java

1. create a product data object

For product SOFA1

column Name

column data

```
public static void main(String[] args){
    addData();
}

static void addData(){
    ProductPersistenceHandler persistenceHandler = new
ProductPersistenceHandler();
    Product product1 = new Product();
    Map<String, Object> attributesMaps1 = Maps.newHashMap();
    attributesMaps1.put(CATEGORY.getValue(), "sofa");
    attributesMaps1.put(BRAND.getValue(), "Fab");
    attributesMaps1.put(BREADTH.getValue(), 100);
    attributesMaps1.put(HEIGHT.getValue(), 200);
    attributesMaps1.put(LENGTH.getValue(), 500);
    attributesMaps1.put(TITLE.getValue(), "Urban Living Derby");
    product1.setProductId("SOFA1");
    product1.setAttributesMap(attributesMaps1);
    persistenceHandler.insertProducts(product1);
}
```

AddProductsMain.java

1. create a product data object

For product SOFA1

we set this attributeMap and productId in product1 object

Our SOFA1 product object is ready

```
public static void main(String[] args){
    addData();
}

static void addData(){
    ProductPersistenceHandler persistenceHandler = new
ProductPersistenceHandler();
    Product product1 = new Product();
    Map<String, Object> attributesMaps1 = Maps.newHashMap();
    attributesMaps1.put(CATEGORY.getValue(), "sofa");
    attributesMaps1.put(BRAND.getValue(), "Fab");
    attributesMaps1.put(BREADTH.getValue(), 100);
    attributesMaps1.put(HEIGHT.getValue(), 200);
    attributesMaps1.put(LENGTH.getValue(), 500);
    attributesMaps1.put(TITLE.getValue(), "Urban Living Derby");
    product1.setProductId("SOFA1");
    product1.setAttributesMap(attributesMaps1);
    persistenceHandler.insertProducts(product1);
}
```

We add products in db in 2 steps

1. create a product data object

2. ProductPersistenceHandler inserts the product in db

ProductPersistenceHandler.java

2. ProductPersistenceHandler inserts the product in db

As we did with listing

```
public class ProductPersistenceHandler {
  private static String keyspace = "cms";
  private static String columnFamily = "products";

  public void insertProducts(Product product){


        Insert insertStatement = QueryBuilder.insertInto(keyspace,
columnFamily);

        Map<String, Object> attributes = product.getAttributesMap();
        insertStatement.value(Attrib
product.getProductId());
        for(String attributeName : attributes.keySet()){
            insertStatement = i
attributes.get(attributeName));
        }
        insertStatement.setDefaultTimestamp(new
ThreadLocalMonotonicTimestampGe
        Session session = Connector.getSession();
        session.execute(insertStatem

        session.close();


  }
```

We use insertInto() method of QueryBuilder

keyspace and clumnFamily is defined in the class

ProductPersistenceHandler.java

2. ProductPersistenceHandler inserts the product in db

```
public class ProductPersistenceHandler {
  private static String keyspace = "cms";
  private static String columnFamily = "products";

  public void insertProducts(Product product){



        Insert insertStatement = QueryBuilder.insertInto(keyspace,
columnFamily);

        Map<String, Object> attributes = product.getAttributesMap();
        insertStatement.value(AttributeNames.PRODUCTID.getValue(),
product.getProductId());
        for(String attributeName : attributes.keySet()){
            insertStatement = insertStatement.value(attributeName,
attributes.get(attributeName));
        }
        insertStatement.setDefaultTimestamp(new
ThreadLocalMonotonicTimestampGenerator().next());
        Session session = Connector.getSession();
        session.execute(insertStatement);

        session.close();

  }
```

By using value() we add the attribute data to the insert statement

ProductPersistenceHandler.java

2. ProductPersistenceHandler inserts the product in db

```
public class ProductPersistenceHandler {
 private static String keyspace = "cms";
 private static String columnFamily = "products";

 public void insertProducts(Product product){


    Insert insertStatement = QueryBuilder.insertInto(keyspace,
columnFamily);

    Map<String, Object> attributes = product.getAttributesMap();
    insertStatement.value(AttributeNames.PRODUCTID.getValue(),
product.getProductId());
    for(String attributeName : attributes.keySet()){
      insertStatement = insertStatement.value(attributeName,
attributes.get(attributeName));
    }
    insertStatement.setDefaultTimestamp(new
ThreadLocalMonotonicTimestampGenerator().next());
    Session session = Connector.getSession();
    session.execute(insertStatement);

    session.close();

 }
```

we set the timestamp using setDefaultTimeStamp

ProductPersistenceHandler.java

2. ProductPersistenceHandler inserts the product in db

```
public class ProductPersistenceHandler {
  private static String keyspace = "cms";
  private static String columnFamily = "products";

  public void insertProducts(Product product){


        Insert insertStatement = QueryBuilder.insertInto(keyspace,
columnFamily);

        Map<String, Object> attributes = product.getAttributesMap();
        insertStatement.value(AttributeNames.PRODUCTID.getValue(),
product.getProductId());
        for(String attributeName : attributes.keySet()){
            insertStatement = insertStatement.value(attributeName,
attributes.get(attributeName));
        }
        insertStatement.setDefaultTimestamp(new
ThreadLocalMonotonicTimestampGenerator().next());
        Session session = Connector.getSession();
        session.execute(insertStatement);

        session.close();

  }
```

get the session object and execute the statement

We add products in db in 2 steps

1. create a product data object

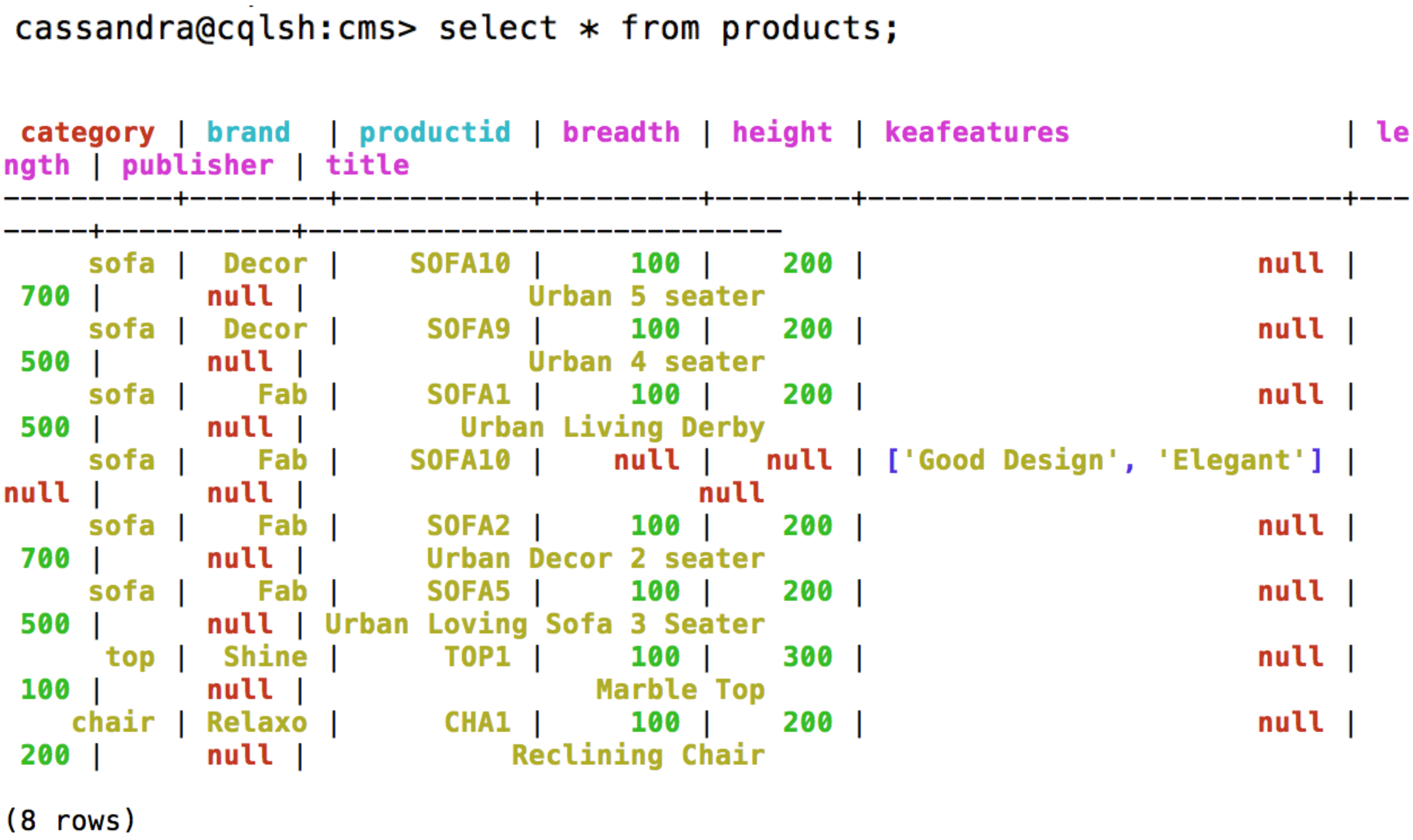
2. ProductPersistenceHandler inserts the product in db

In the same way we have added 8 furniture products in the db

You can add products by running main() of AddProductsMain.java

Let's see the products that we have added

Products in db



```
cassandra@cqlsh:cms> select * from products;

 category | brand  | productid | breadth | height | keafeatures                 | le
ngth | publisher | title
----------+--------+-----------+---------+--------+-----------------------------+---
-----+-----------+----------------------------
     sofa |  Decor |    SOFA10 |     100 |    200 |                        null |
 700 |      null |             Urban 5 seater
     sofa |  Decor |     SOFA9 |     100 |    200 |                        null |
 500 |      null |             Urban 4 seater
     sofa |    Fab |     SOFA1 |     100 |    200 |                        null |
 500 |      null |         Urban Living Derby
     sofa |    Fab |    SOFA10 |    null |   null |  ['Good Design', 'Elegant'] |
null |      null |                       null
     sofa |    Fab |     SOFA2 |     100 |    200 |                        null |
 700 |      null |       Urban Decor 2 seater
     sofa |    Fab |     SOFA5 |     100 |    200 |                        null |
 500 |      null | Urban Loving Sofa 3 Seater
      top |  Shine |      TOP1 |     100 |    300 |                        null |
 100 |      null |                 Marble Top
    chair | Relaxo |      CHA1 |     100 |    200 |                        null |
 200 |      null |            Reclining Chair

(8 rows)
```

Now that we have the data set, let's search for the following products

products with the "Fab" brand in categories sofa and chair

We use 3 classes

Product class - the structure to represent the product data

ProductPersistenceHandler class - the class to query db and get the required data

Main class - has the input parameters

Product.java Class

```java
public class Product {

    String productId;

    Map<String, Object> attributesMap;

    public String getProductId() {
        return productId;
    }

    public void setProductId(String productId) {
        this.productId = productId;
    }

    public Map<String, Object> getAttributesMap() {
        return attributesMap;
    }


    public void setAttributesMap(Map<String, Object> attributesMap) {
        this.attributesMap = attributesMap;
    }


    @Override
    public String toString() {
        return "Product{" +
                "productId='" + productId + '\'' +
                ", attributesMap=" + attributesMap +
                '}';
    }
}
```

This class is used to represent the product data

It has a productId and attributesMap

These are getter and setter methods

Product.java Class

```java
public class Product {

    String productId;

    Map<String, Object> attributesMap;

    public String getProductId() {
        return productId;
    }

    public void setProductId(String productId) {
        this.productId = productId;
    }

    public Map<String, Object> getAttributesMap() {
        return attributesMap;
    }


    public void setAttributesMap(Map<String, Object> attributesMap) {
        this.attributesMap = attributesMap;
    }



    @Override
    public String toString() {
        return "Product{" +
                "productId='" + productId + '\'' +
                ", attributesMap=" + attributesMap +
                '}';
    }
```

This class is used to represent the product data

AttributesMap stores attribute data in the form of key->value pair

attribute name is the key

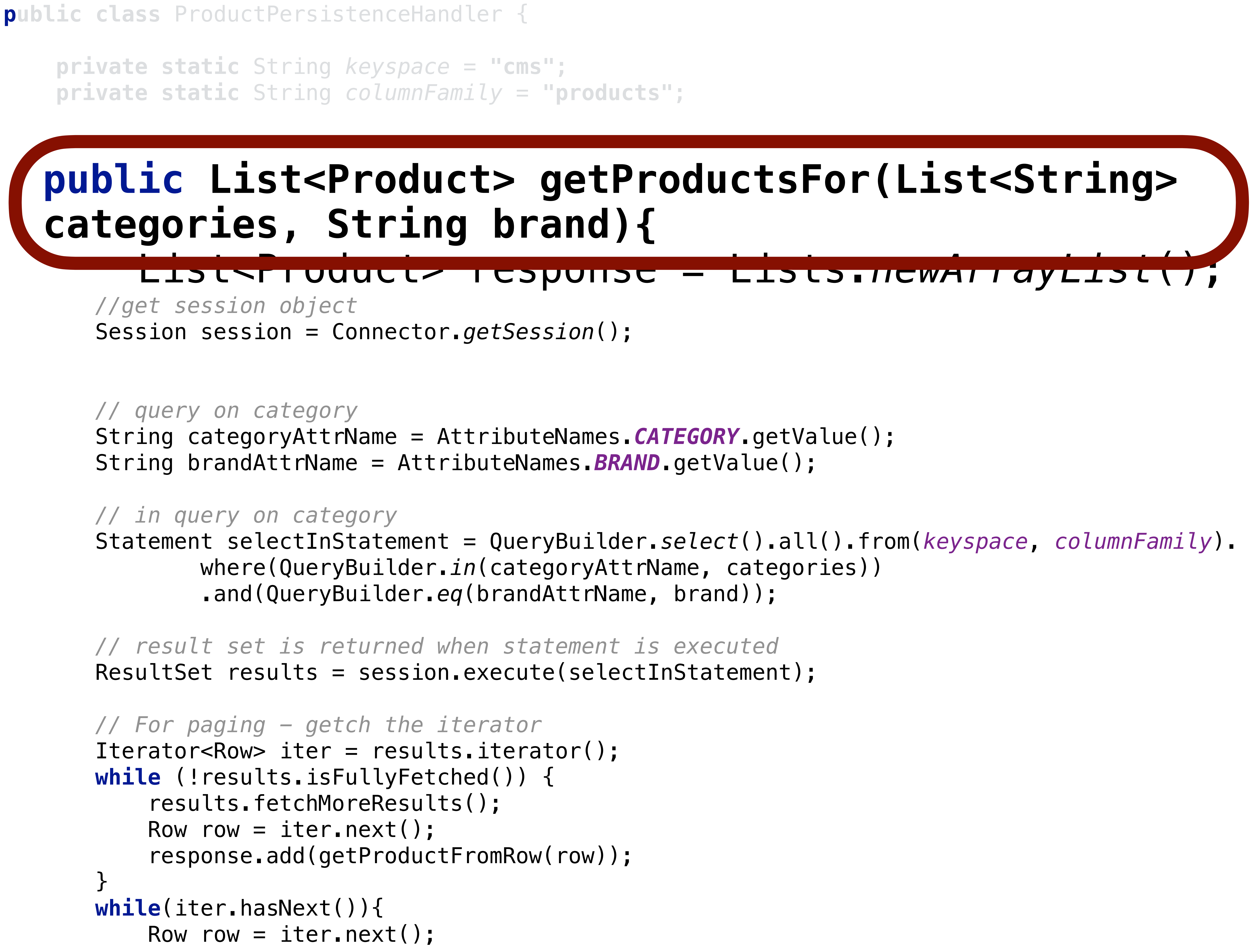
attribute data is the value

ProductPersistenceHandler Class

```
public class ProductPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = "products";


   public List<Product> getProductsFor(List<String> categories, String brand){
       List<Product> response = Lists.newArrayList();
       Session session = Connector.getSession();


       String categoryAttrName = AttributeNames.CATEGORY.getValue();
       String brandAttrName = AttributeNames.BRAND.getValue();


       Statement selectInStatement = QueryBuilder.select().all().from(keyspace, columnFamily).
               where(QueryBuilder.in(categoryAttrName, categories))
               .and(QueryBuilder.eq(brandAttrName, brand));


       ResultSet results = session.execute(selectInStatement);


       Iterator<Row> iter = results.iterator();
       while (!results.isFullyFetched()) {
           results.fetchMoreResults();
           Row row = iter.next();
           response.add(getProductFromRow(row));
       }
       while(iter.hasNext()){
           Row row = iter.next();
           response.add(getProductFromRow(row));
       }
       return response;

   }
```

ProductPersistenceHandler
interacts with cassandra
for all product CF operations

ProductPersistenceHandler Class



```
public class ProductPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = "products";

public List<Product> getProductsFor(List<String>
categories, String brand){
        List<Product> response = Lists.newArrayList();
        //get session object
        Session session = Connector.getSession();


        // query on category
        String categoryAttrName = AttributeNames.CATEGORY.getValue();
        String brandAttrName = AttributeNames.BRAND.getValue();

        // in query on category
        Statement selectInStatement = QueryBuilder.select().all().from(keyspace, columnFamily).
                where(QueryBuilder.in(categoryAttrName, categories))
                .and(QueryBuilder.eq(brandAttrName, brand));

        // result set is returned when statement is executed
        ResultSet results = session.execute(selectInStatement);

        // For paging - getch the iterator
        Iterator<Row> iter = results.iterator();
        while (!results.isFullyFetched()) {
            results.fetchMoreResults();
            Row row = iter.next();
            response.add(getProductFromRow(row));
        }
        while(iter.hasNext()){
            Row row = iter.next();
```

getProductsFor() is the method we will use to the required products

ProductPersistenceHandler Class

```
public class ProductPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = "products";

   public List<Product> getProductsFor(List<String> categories, String brand){
       List<Product> response = Lists.newArrayList();
      //get session object
      Session session = Connector.getSession();


      // query on category
      String categoryAttrName = AttributeNames.CATEGORY.getValue();
      String brandAttrName = AttributeNames.BRAND.getValue();

      // in query on category
      Statement selectInStatement = QueryBuilder.select().all().from(keyspace, columnFamily).
              where(QueryBuilder.in(categoryAttrName, categories))
              .and(QueryBuilder.eq(brandAttrName, brand));

      // result set is returned when statement is executed
      ResultSet results = session.execute(selectInStatement);

      // For paging - getch the iterator
      Iterator<Row> iter = results.iterator();
      while (!results.isFullyFetched()) {
          results.fetchMoreResults();
          Row row = iter.next();
          response.add(getProductFromRow(row));
      }
      while(iter.hasNext()){
          Row row = iter.next();
          response.add(getProductFromRow(row));
      }
```

It takes a list of categories and a brand as arguments

ProductPersistenceHandler Class

```
public class ProductPersistenceHandler {
private Product getProductFromRow(Row row){
    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if(row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions =
defns.asList();
        for (ColumnDefinitions.Definition columnDefn : columnDefinitions){
            String columnName = columnDefn.getName();
            Object data = row.getObject(columnName);
            if (AttributeNames.PRODUCTID.getValue().equals(columnName)) {
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }

    }
    product.setAttributesMap(attributes);
    return product;
  }
}
```

getProductsFromRow transforms the row returned from the database into a Product object

ProductPersistenceHandler Class

```
public class ProductPersistenceHandler {

private Product getProductFromRow(Row row){

    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if (row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions =
defns.asList();
        for (ColumnDefinitions.Definition columnDefn : columnDefinitions){
            String columnName = columnDefn.getName();
            Object data = row.getObject(columnName);
            if (AttributeNames.PRODUCTID.getValue().equals(columnName)) {
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }

    }
    product.setAttributesMap(attributes);
    return product;
  }
}
```

return Product

input row

Let's see the steps we will follow

1. prepare parameters for search
2. make prepared statement
3. execute the statement
4. Iterate over the result set to return set of products

Main.java Class

1. prepare parameters for search

Main class

We will search for products in the sofa or chair categories with brand Fab

```java
public class Main {

public static void main(String[] args){
{
   getProducts();

}

static  void getProducts(){
    List<String> categories = Lists.newArrayList();
    categories.add("sofa");
    categories.add("chair");

    String brand = "Fab";

    ProductPersistenceHandler handler = new
ProductPersistenceHandler();
    List<Product> products = handler.getProductsFor(categories,
brand);

    System.out.println(products);
}

}
```

Main.java Class

1. prepare parameters for search

Main class

```
public class Main {

public static void main(String[] args){
{
   getProducts();

}

static  void getProducts(){
    List<String> categories = Lists.newArrayList();
    categories.add("sofa");
    categories.add("chair");

    String brand = "Fab";

    ProductPersistenceHandler handler = new
ProductPersistenceHandler();
    List<Product> products = handler.getProductsFor(categories,
brand);

    System.out.println(products);
}

}
```

we call the productPersistenceHandler to get the products matching our parameters

[sofa, chair]
Fab

Main.java Class

1. prepare parameters for search

Main class

```
public class Main {

public static void main(String[] args){
{
   getProducts();

}

static  void getProducts(){
    List<String> categories = Lists.newArrayList();
    categories.add("sofa");
    categories.add("chair");

    String brand = "Fab";

    ProductPersistenceHandler handler = new
ProductPersistenceHandler();
    List<Product> products = handler.getProductsFor(categories,
brand);

    System.out.println(products);
}

}
```

productPersistenceHandler will return list of products matching the criteria

Let's see the steps we will follow

1. prepare parameters for search
2. **make prepared statement**
3. execute the statement
4. Iterate over the result set to return set of products

ProductPersistenceHandler Class

2. make prepared statement

```
public class ProductPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = "products";


   public List<Product> getProductsFor(List<String> categories, String brand){
       List<Product> response = Lists.newArrayList();
       Session session = Connector.getSession();


       String categoryAttrName = AttributeNames.CATEGORY.getValue();
       String brandAttrName = AttributeNames.BRAND.getValue();


       Statement selectInStatement = QueryBuilder.select().all().from(keyspace, columnFamily).
               where(QueryBuilder.in(categoryAttrName, categories))
               .and(QueryBuilder.eq(brandAttrName, brand));


       ResultSet results = session.execute(selectInStatement);


       Iterator<Row> iter = results.iterator();
       while (!results.isFullyFetched()) {
           results.fetchMoreResults();
           Row row = iter.next();
           response.add(getProductFromRow(row));
       }
       while(iter.hasNext()){
           Row row = iter.next();
           response.add(getProductFromRow(row));
       }
       return response;

   }
```

ProductPersistenceHandler Class

2. make prepared statement

```
public class ProductPersistenceHandler {

    private static String keyspace = "cms";
      private static String columnFamily = "products";

   public List<Product> getProductsFor(List<String> categories, String brand){
       List<Product> response = Lists.newArrayList();
       Session session = Connector.getSession();

       String categoryAttrName = AttributeNames.CATEGORY.getValue();
       String brandAttrName = AttributeNames.BRAND.getValue();

       Statement selectInStatement = QueryBuilder.select().all().from(keyspace, columnFamily).
               where(QueryBuilder.in(categoryAttrName, categories))
               .and(QueryBuilder.eq(brandAttrName, brand));

       ResultSet results = session.execute(selectInStatement);

       Iterator<Row> iter = results.iterator();
       while (!results.isFullyFetched()) {
           results.fetchMoreResults();
           Row row = iter.next();
           response.add(getProductFromRow(row));
       }
       while(iter.hasNext()){
           Row row = iter.next();
           response.add(getProductFromRow(row));
       }
       return response;
```

first we get the session object

ProductPersistenceHandler Class

2. make prepared statement

We get the column names for category and brand from the AttributeNames enum

```
ublic class ProductPersistenceHandler {

   private static String keyspace = "cms";
    private static String columnFamily = "products";

  public List<Product> getProductsFor(List<String> categories, String brand){
      List<Product> response = Lists.newArrayList();
      Session session = Connector.getSession();

      String categoryAttrName = AttributeNames.CATEGORY.getValue();
        String brandAttrName = AttributeNames.BRAND.getValue();

      Statement selectInStatement = QueryBuilder.select().all().from(keyspace, columnFamily).
              where(QueryBuilder.in(categoryAttrName, categories))
              .and(QueryBuilder.eq(brandAttrName, brand));

      ResultSet results = session.execute(selectInStatement);

      Iterator<Row> iter = results.iterator();
      while (!results.isFullyFetched()) {
          results.fetchMoreResults();
          Row row = iter.next();
          response.add(getProductFromRow(row));
      }
      while(iter.hasNext()){
          Row row = iter.next();
          response.add(getProductFromRow(row));
      }
```

categoryAttrName has the column name for category

brandAttrName has the column name for brand

ProductPersistenceHandler Class

2. make prepared statement

```
public class ProductPersistenceHandler {

   private static String keyspace = "cms";
     private static String columnFamily = "products";

  public List<Product> getProductsFor(List<String> categories, String brand){
     List<Product> response = Lists.newArrayList();
     Session session = Connector.getSession();

     String categoryAttrName = AttributeNames.CATEGORY.getValue();
     String brandAttrName = AttributeNames.BRAND.getValue();


       Statement selectInStatement =
               QueryBuilder.select().all().from(keyspace, columnFamily).
               where(QueryBuilder.in(categoryAttrName, categories))
               .and(QueryBuilder.eq(brandAttrName, brand));

     ResultSet results = session.execute(selectInStatement);

     Iterator<Row> iter = results.iterator();
     while (!results.isFullyFetched()) {
         results.fetchMoreResults();
         Row row = iter.next();
         response.add(getProductFromRow(row));
     }
     while(iter.hasNext()){
         Row row = iter.next();
         response.add(getProductFromRow(row));
```

select

Query Builder has methods for all cql query operators

select() returns selection object to start building a select query

ProductPersistenceHandler Class

2. make prepared statement

```
public class ProductPersistenceHandler {

  private static String keyspace = "cms";
   private static String columnFamily = "products";

 public List<Product> getProductsFor(List<String> categories, String brand){
    List<Product> response = Lists.newArrayList();
    Session session = Connector.getSession();

    String categoryAttrName = AttributeNames.CATEGORY.getValue();
    String brandAttrName = AttributeNames.BRAND.getValue();

    Statement selectInStatement =
            QueryBuilder.select().all().from(keyspace, columnFamily).
             where(QueryBuilder.in(categoryAttrName, categories))
             .and(QueryBuilder.eq(brandAttrName, brand));

    ResultSet results = session.execute(selectInStatement);

    Iterator<Row> iter = results.iterator();
    while (!results.isFullyFetched()) {
        results.fetchMoreResults();
        Row row = iter.next();
        response.add(getProductFromRow(row));
    }
    while(iter.hasNext()){
        Row row = iter.next();
        response.add(getProductFromRow(row));
```

select *

all() builder selects all the columns in the result

it returns a partially built Select statement

ProductPersistenceHandler Class

2. make prepared statement

```
public class ProductPersistenceHandler {

  private static String keyspace = "cms";
    private static String columnFamily = "products";


  public List<Product> getProductsFor(List<String> categories, String brand){
      List<Product> response = Lists.newArrayList();
      Session session = Connector.getSession();

      String categoryAttrName = AttributeNames.CATEGORY.getValue();
      String brandAttrName = AttributeNames.BRAND.getValue();


      Statement selectInStatement =
              QueryBuilder.select().all().from(keyspace, columnFamily).
               where(QueryBuilder.in(categoryAttrName, categories))
               .and(QueryBuilder.eq(brandAttrName, brand));

      ResultSet results = session.execute(selectInStatement);


      Iterator<Row> iter = results.iterator();
      while (!results.isFullyFetched()) {
          results.fetchMoreResults();
          Row row = iter.next();
          response.add(getProductFromRow(row));
      }
      while(iter.hasNext()) {
          Row row = iter.next();
          response.add(getProductFromRow(row));
```

select * from cms.product

from() in Select adds keyspace and columnfamily to query

executable in-build SELECT statement is returned

ProductPersistenceHandler Class

2. make prepared statement

```
public class ProductPersistenceHandler {

  private static String keyspace = "cms";
    private static String columnFamily = "products";


  public List<Product> getProductsFor(List<String> categories, String brand){
      List<Product> response = Lists.newArrayList();
      Session session = Connector.getSession();
```

```
select * from cms.product where category in (‘sofa’, ‘chair’)
```

```
      String categoryAttrName = AttributeNames.CATEGORY.getValue();
      String brandAttrName = AttributeNames.BRAND.getValue();


      Statement selectInStatement =
              QueryBuilder.select().all().from(keyspace, columnFamily).
              where(QueryBuilder.in(categoryAttrName, categories))
              .and(QueryBuilder.eq(brandAttrName, brand));


      ResultSet results = session.execute(selectInStatement);


      Iterator<Row> iter = results.iterator();
      while (!results.isFullyFetched()) {
          results.fetchMoreResults();
          Row row = iter.next();
          response.add(getProductFromRow(row));
      }
      while(iter.hasNext()){
          Row row = iter.next();
          response.add(getProductFromRow(row));
```

1st parameter is the attribute name

Use IN method of QueryBuilder for categories

2nd parameter is the attribute value list

ProductPersistenceHandler Class

2. make prepared statement

```
public class ProductPersistenceHandler {

  private static String keyspace = "cms";
    private static String columnFamily = "products";


 public List<Product> getProductsFor(List<String> categories, String brand){
     List<Product> response = Lists.newArrayList();
     Session session = Connector.getSession();


     String categoryAttrName = AttributeNames.CATEGORY.getValue();
     String brandAttrName = AttributeNames.BRAND.getValue();
```

select * from cms.product where category in (‘sofa’, ‘chair’) and brand = ‘Fab’;

```
    Statement selectInStatement =
            QueryBuilder.select().all().from(keyspace, columnFamily).
             where(QueryBuilder.in(categoryAttrName, categories))
             .and(QueryBuilder.eq(brandAttrName, brand));

     ResultSet results = session.execute(selectInStatement);


     Iterator<Row> iter = results.iterator();
     while (!results.isFullyFetched()) {
         results.fetchMoreResults();
         Row row = iter.next();
         response.add(getProductFromRow(row));
     }
     while(iter.hasNext()){
         Row row = iter.next();
         response.add(getProductFromRow(row));
```

Use EQ method of QueryBuilder for = operation on brand

with this we form the full select statement

Let's see the steps we will follow

1. prepare parameters for search
2. make prepared statement
3. **execute the statement**
4. Iterate over the result set to return set of products

ProductPersistenceHandler Class

3. execute the statement

```
public class ProductPersistenceHandler {

    private static String keyspace = "cms";
      private static String columnFamily = "products";


   public List<Product> getProductsFor(List<String> categories, String brand){
       List<Product> response = Lists.newArrayList();
       Session session = Connector.getSession();


       String categoryAttrName = AttributeNames.CATEGORY.getValue();
       String brandAttrName = AttributeNames.BRAND.getValue();




         Statement selectInStatement =
              QueryBuilder.select().all().from(keyspace, columnFamily).
               where(QueryBuilder.in(categoryAttrName, categories))
               .and(QueryBuilder.eq(brandAttrName, brand));


         ResultSet results = session.execute(selectInStatement);


       Iterator<Row> iter = results.iterator();
       while (!results.isFullyFetched()) {
           results.fetchMoreResults();
           Row row = iter.next();
           response.add(getProductFromRow(row));
       }
       while(iter.hasNext()){
           Row row = iter.next();
           response.add(getProductFromRow(row));
       }
```

execute() returns resultSet which holds the result of the query

Let's see the steps we will follow

1. prepare parameters for search
2. make prepared statement
3. execute the statement
4. Iterate over the result set to return set of products

ProductPersistenceHandler Class

```
 private static String keyspace = "cms";
   private static String columnFamily = "products";


public List<Product> getProductsFor(List<String> categories, String brand){
    List<Product> response = Lists.newArrayList();
    Session session = Connector.getSession();


    String categoryAttrName = AttributeNames.CATEGORY.getValue();
    String brandAttrName = AttributeNames.BRAND.getValue();



      Statement selectInStatement =
           QueryBuilder.select().all().from(keyspace, columnFamily).
            where(QueryBuilder.in(categoryAttrName, categories))
            .and(QueryBuilder.eq(brandAttrName, brand));


     ResultSet results = session.execute(selectInStatement);


    Iterator<Row> iter = results.iterator();
    while(!results.isFullyFetched()){
        results.fetchMoreResults();
        Row row = iter.next();
        response.add(getProductFromRow(row));
    }
      while(iter.hasNext()){
            Row row = iter.next();
            response.add(getProductFromRow(row));
    }
    return response;

}
```

4. Iterate over the result set to return set of products

result set returns the first page of the result

result set provides an iterator to consume the rows

it fetches the next page when the results of the first page are consumed

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```
private static String keyspace = "cms";
  private static String columnFamily = "products";


public List<Product> getProductsFor(List<String> categories, String brand){
    List<Product> response = Lists.newArrayList();
    Session session = Connector.getSession();


    String categoryAttrName = AttributeNames.CATEGORY.getValue();
    String brandAttrName = AttributeNames.BRAND.getValue();



      Statement selectInStatement =
           QueryBuilder.select().all().from(keyspace, columnFamily).
            where(QueryBuilder.in(categoryAttrName, categories))
            .and(QueryBuilder.eq(brandAttrName, brand));


     ResultSet results = session.execute(selectInStatement);


     Iterator<Row> iter = results.iterator();
    while (!results.isFullyFetched()) {
          results.fetchMoreResults();
          Row row = iter.next();
          response.add(getProductFromRow(row));
     }
     while(iter.hasNext()){
          Row row = iter.next();
          response.add(getProductFromRow(row));
    }
```

we can prefetch pages as well

fetchMoreResults() fetches all pages before hand

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```
private static String keyspace = "cms";
  private static String columnFamily = "products";


public List<Product> getProductsFor(List<String> categories, String brand){
    List<Product> response = Lists.newArrayList();
    Session session = Connector.getSession();


    String categoryAttrName = AttributeNames.CATEGORY.getValue();
    String brandAttrName = AttributeNames.BRAND.getValue();



      Statement selectInStatement =
           QueryBuilder.select().all().from(keyspace, columnFamily).
            where(QueryBuilder.in(categoryAttrName, categories))
            .and(QueryBuilder.eq(brandAttrName, brand));


     ResultSet results = session.execute(selectInStatement);


     Iterator<Row> iter = results.iterator();
    while (!results.isFullyFetched()) {
             results.fetchMoreResults();
             Row row = iter.next();
             response.add(getProductFromRow(row));
      }
      while(iter.hasNext()){
             Row row = iter.next();
             response.add(getProductFromRow(row));
    }
```

row contains the current row data and the information of its columns

getProductFromRow transforms row data into our Product object

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```
public class ProductPersistenceHandler {

private Product getProductFromRow(Row row)
{

    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if(row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions =
defns.asList();
        for (ColumnDefinitions.Definition columnDefn :
columnDefinitions){
            String columnName = columnDefn.getName();
            Object data = row.getObject(columnName);
            if
(AttributeNames.PRODUCTID.getValue().equals(columnName)) {
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }

    }
    product.setAttributesMap(attributes);
    return product;
```

getProductFromRow transform the row into Product data

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```
public class ProductPersistenceHandler {

private Product getProductFromRow(Row row){

    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if(row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions
= defns.asList();
        for (ColumnDefinitions.Definition columnDefn : columnDefinitions){
            String columnName = columnDefn.getName();
            Object data = row.getObject(columnName);
            if (AttributeNames.PRODUCT_ID.getValue().equals(columnName)){
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }

    }
    product.setAttributesMap(attributes);
    return product;
}
```

column definitions contains information only for the columns present in the row

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

each column definition contains the information about the name, datatype, keyspace, CF the column belongs to etc.

```
public class ProductPersistenceHandler {

private Product getProductFromRow(Row row){

    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if(row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions = defns.asList();
     for (ColumnDefinitions.Definition columnDefn :
columnDefinitions){
            String columnName = columnDefn.getName();
            Object data = row.getObject(columnName);
            if (AttributeNames.PRODUCTID.getValue().equals(columnName)) {
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }

    }
    product.setAttributesMap(attributes);
```

we iterate over these definitions

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```java
public class ProductPersistenceHandler {

private Product getProductFromRow(Row row){

    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if(row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions = defns.asList();
        for (ColumnDefinitions.Definition columnDefn :
columnDefinitions){
                         String columnName = columnDefn.getName();
                         Object data = row.getObject(columnName);
            if (AttributeNames.PRODUCTID.getValue().equals(columnName)) {
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }
    }
    product.setAttributesMap(attributes);
    return product;
```

get column name from the definition and get the corresponding value from row

values of columns in row can be accessed by either index or by name

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```
public class ProductPersistenceHandler {

private Product getProductFromRow(Row row){

    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if(row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions = defns.asList();
        for (ColumnDefinitions.Definition columnDefn :
columnDefinitions){
                String columnName = columnDefn.getName();
                Object data = row.getObject(columnName);
            if (AttributeNames.PRODUCTID.getValue().equals(columnName)) {
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }
    }
    product.setAttributesMap(attributes);
    return product;
```

row can return value as String, bool, int and other data types

getObject() returns in the java type that is equivalent to cql type

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```
public class ProductPersistenceHandler {

private Product getProductFromRow(Row row){

    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if(row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions =
defns.asList();
        for (ColumnDefinitions.Definition columnDefn : columnDefinitions)
{
            String columnName = columnDefn.getName();
            Object data = row.getObject(columnName);
            if (AttributeNames.PRODUCTID.getValue().equals(columnName)) {
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }

    }
    product.setAttributesMap(attributes);
    return product;
}
```

productId is a separate field in the product object

we put the remaining column data in attributes map

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```
public class ProductPersistenceHandler {

private Product getProductFromRow(Row row){

    Product product = new Product();
    Map<String, Object> attributes = Maps.newHashMap();

    if(row != null) {
        ColumnDefinitions defns = row.getColumnDefinitions();
        List<ColumnDefinitions.Definition> columnDefinitions =
defns.asList();
        for (ColumnDefinitions.Definition columnDefn :
columnDefinitions){
            String columnName = columnDefn.getName();
            Object data = row.getObject(columnName);
            if
(AttributeNames.PRODUCTID.getValue().equals(columnName)) {
                product.setProductId(data.toString());
            }else{
                attributes.put(columnName, data);
            }
        }

    }

product.setAttributesMap(attributes);
    return product;
```

we set the attributes in product object and return the product object

we do this for all rows returned in the result

ProductPersistenceHandler Class

4. Iterate over the result set to return set of products

```
 private static String keyspace = "cms";
   private static String columnFamily = "products";

public List<Product> getProductsFor(List<String> categories,
String brand){
    List<Product> response = Lists.newArrayList();
    Session session = Connector.getSession();


    String categoryAttrName = AttributeNames.CATEGORY.getValue();
    String brandAttrName = AttributeNames.BRAND.getValue();



      Statement selectInStatement =
          QueryBuilder.select().all().from(keyspace, columnFamily).
           where(QueryBuilder.in(categoryAttrName, categories))
           .and(QueryBuilder.eq(brandAttrName, brand));


    ResultSet results = session.execute(selectInStatement);


    Iterator<Row> iter = results.iterator();
    while (!results.isFullyFetched()) {
        results.fetchMoreResults();
        Row row = iter.next();
        response.add(getProductFromRow(row));
    }
    while(iter.hasNext()){
            Row row = iter.next();
        response.add(getProductFromRow(row));
    }
    return response;
```

getProductsFor() returns the list of products to the Main class

Let's see the steps we will follow

1. prepare parameters for search
2. make prepared statement
3. execute the statement
4. Iterate over the result set to return set of products

Main.java Class

Main class

```
public class Main {

public static void main(String[] args){
{
   getProducts();

}

static  void getProducts(){
    List<String> categories = Lists.newArrayList();
    categories.add("sofa");
    categories.add("chair");

    String brand = "Fab";

    ProductPersistenceHandler handler = new ProductPersistenceHandler();
    List<Product> products = handler.getProductsFor(categories, brand);

    System.out.println(products);
}

}
```

print the fetched products

Output

```
[Product{productId='SOFA1', attributesMap={breadth=100,
length=500, publisher=null, category=sofa, title=Urban
Living Derby, brand=Fab, height=200, keafeatures=[]}},

Product{productId='SOFA10', attributesMap={breadth=null,
length=null, publisher=null, category=sofa, title=null,
brand=Fab, height=null, keafeatures=[Good Design,
Elegant]}},

Product{productId='SOFA2', attributesMap={breadth=100,
length=700, publisher=null, category=sofa, title=Urban
Decor 2 seater, brand=Fab, height=200, keafeatures=[]}},

Product{productId='SOFA5', attributesMap={breadth=100,
length=500, publisher=null, category=sofa, title=Urban
Loving Sofa 3 Seater, brand=Fab, height=200,
keafeatures=[]}}, Product{productId='null',
attributesMap={}}]
```

Let’s say that seller decides to delist a product

We need to delete the listing for the product

Let's delete listings for products CHA1 and TOP1

We will use 2 classes here

ListingPersistenceHandler class has the method delete() to delete listings

Main class calls the delete() with input parameters

Let's see the steps we will follow

1. make delete prepared statement
2. get session object
3. execute the statement

Let's see the steps we will follow

1. make delete prepared statement
2. get session object
3. execute the statement

Main.java Class

1. make delete prepared statement

```java
public class Main {

public static void main(String[] args){
{
  deleteListings();

}

static void deleteListings(){
    List<String> productIds = Lists.newArrayList();
    productIds.add("CHA1");
    productIds.add("TOP1");

    ListingPersistenceHandler handler = new
ListingPersistenceHandler();
    handler.delete(productIds);
}

}
```

list of productids to be deleted

Main.java Class

1. make delete prepared statement

```
public class Main {

public static void main(String[] args){
{
  deleteListings();

}

static void deleteListings(){
    List<String> productIds = Lists.newArrayList();
    productIds.add("CHA1");
    productIds.add("TOP1");

    ListingPersistenceHandler handler = new
ListingPersistenceHandler();
    handler.delete(productIds);
}

}
```

call delete() of handler with the list

ListingPersistenceHandler Class

1. make delete prepared statement

accepts list of products as input

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = “listings";

    public void delete(List<String> productIds){

       String productIdAttrName = AttributeNames.PRODUCTID.getValue();

       Statement deleteStatement = QueryBuilder.delete().from(keyspace, columnFamily)
                    .where(QueryBuilder. in(productIdAttrName,productIds));

       Session session = Connector.getSession();

       session.execute(deleteStatement);

   }
}
```

We can delete by product ids as the product id is the partition key for listings

ListingPersistenceHandler Class

1. make delete prepared statement

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = “listings";


    public void delete(List<String> productIds){

       String productIdAttrName = AttributeNames.PRODUCTID.getValue();

       Statement deleteStatement = QueryBuilder.delete().from(keyspace, columnFamily)
                    .where(QueryBuilder. in(productIdAttrName,productIds));

       Session session = Connector.getSession();

       session.execute(deleteStatement);

   }
}
```

Use the Enum AttributeNamesto get the column names

productIdAttrName stores the column name “productid"

ListingPersistenceHandler Class

1. make delete prepared statement

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = “listings";


    public void delete(List<String> productIds){

       String productIdAttrName = AttributeNames.PRODUCTID.getValue();

       Statement deleteStatement =
QueryBuilder.delete().from(keyspace, columnFamily)
                        .where(
              QueryBuilder.in(productIdAttrName,productIds));

       Session session = Connector.getSession();

       session.execute(deleteStatement);

  }
}
```

delete

we will use the delete() of QueryBuilder to build the query

delete() returns a column selection class for building the delete statement

ListingPersistenceHandler Class

1. make delete prepared statement

delete from cms.listings

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = “listings";


    public void delete(List<String> productIds) {

       String productIdAttrName = AttributeNames.PRODUCTID.getValue();

       Statement deleteStatement =
QueryBuilder.delete().from(keyspace, columnFamily)
                                        .where(
                    QueryBuilder.in(productIdAttrName,productIds));

       Session session = Connector.getSession();

       session.execute(deleteStatement);

  }
}
```

from() adds the keyspace and CF from which we will delete

returns delete statement to delete data from CF

ListingPersistenceHandler Class

1. make delete prepared statement

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = “listings";

      public void delete(List<String> productIds) {

        String productIdAttrName = AttributeNames.PRODUCTID.getValue();

        Statement deleteStatement =
QueryBuilder.delete().from(keyspace, columnFamily)
                        .where(
QueryBuilder.in(productIdAttrName,productIds));

        Session session = Connector.getSession();

        session.execute(deleteStatement);

    }
}
```

delete from cms.listings where productid IN ('TOP1','CHA1');

where clause for conditional delete

productIdAttrName - column Name

productIds - column Value

we use IN method to pass products

Let's see the steps we will follow

1. make delete prepared statement
2. **get session object**
3. execute the statement

ListingPersistenceHandler Class

2. get session object

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = “listings";


    public void delete(List<String> productIds){

       String productIdAttrName = AttributeNames.PRODUCTID.getValue();

       Statement deleteStatement =
QueryBuilder.delete().from(keyspace, columnFamily)
                    .where(
               QueryBuilder.in(productIdAttrName,productIds));

       Session session = Connector.getSession();

       session.execute(deleteStatement);

  }
}
```

get the session object from Connector class

Let's see the steps we will follow

1. make delete prepared statement
2. get session object
3. execute the statement

ListingPersistenceHandler Class

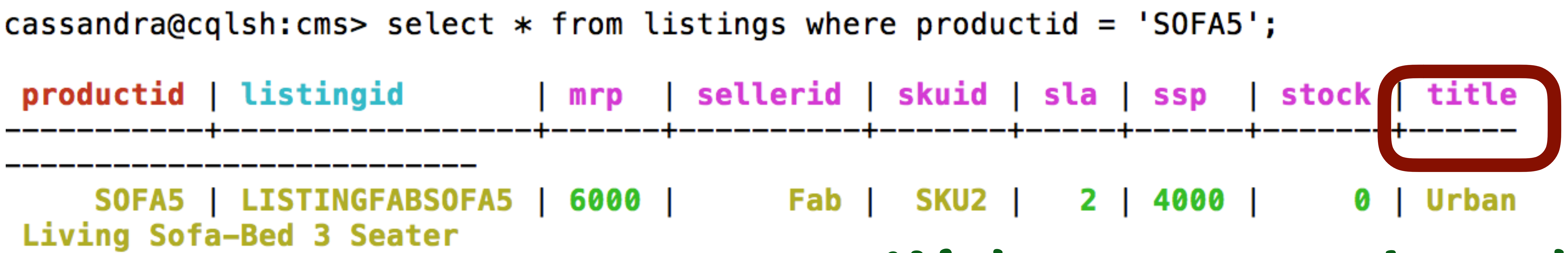
3. execute the statement

```
public class ListingPersistenceHandler {

    private static String keyspace = "cms";
    private static String columnFamily = “listings";


    public void delete(List<String> productIds){

       String productIdAttrName = AttributeNames.PRODUCTID.getValue();

       Statement deleteStatement =
QueryBuilder.delete().from(keyspace, columnFamily)
                           .where(
                     QueryBuilder.in(productIdAttrName,productIds));

       Session session = Connector.getSession();

       session.execute(deleteStatement);

  }
}
```

execute the delete statement

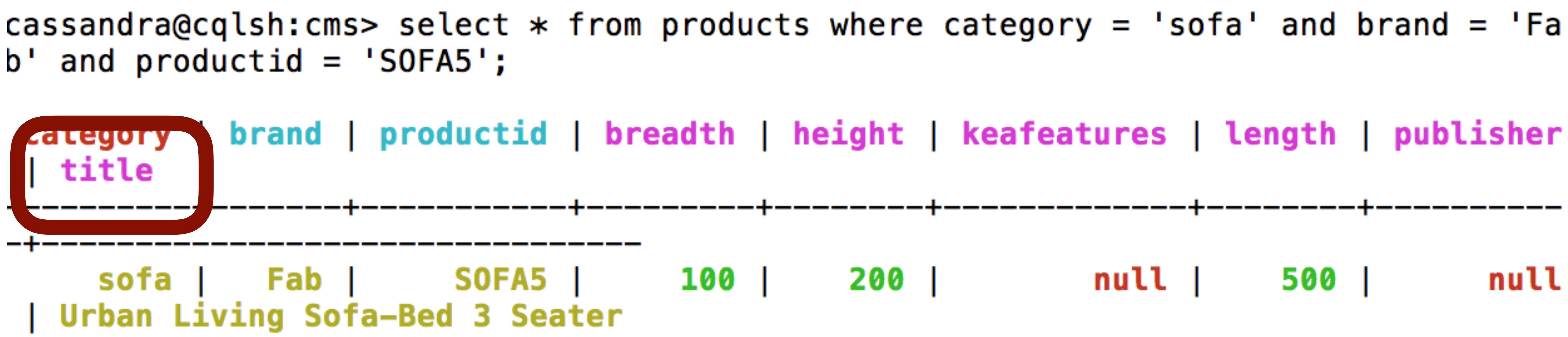
Let's update the product and listing data

Let's have a look at the listing and product data



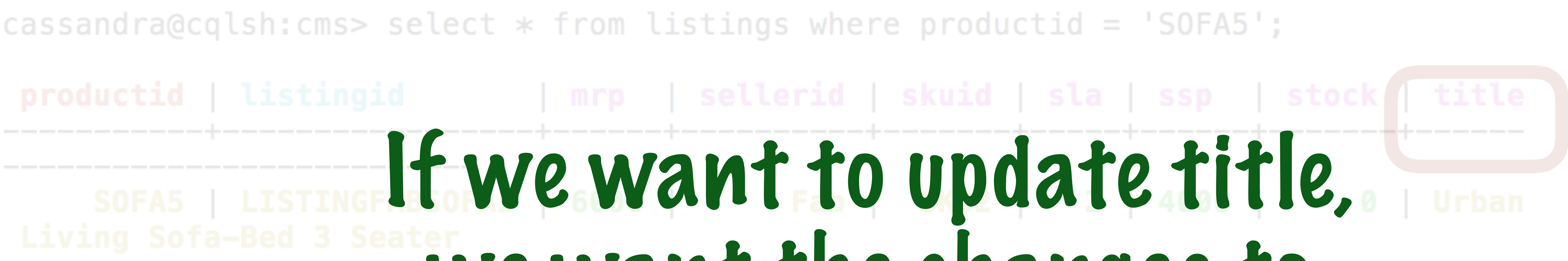
```
cassandra@cqlsh:cms> select * from listings where productid = 'SOFA5';

 productid | listingid         | mrp  | sellerid | skuid | sla | ssp  | stock | title
-----------+-------------------+------+----------+-------+-----+------+-------+-------
---------------------------
     SOFA5 | LISTINGFABSOFA5 | 6000 |      Fab |  SKU2 |   2 | 4000 |     0 | Urban
 Living Sofa-Bed 3 Seater
```

We have denormalised the data added title to listing CF



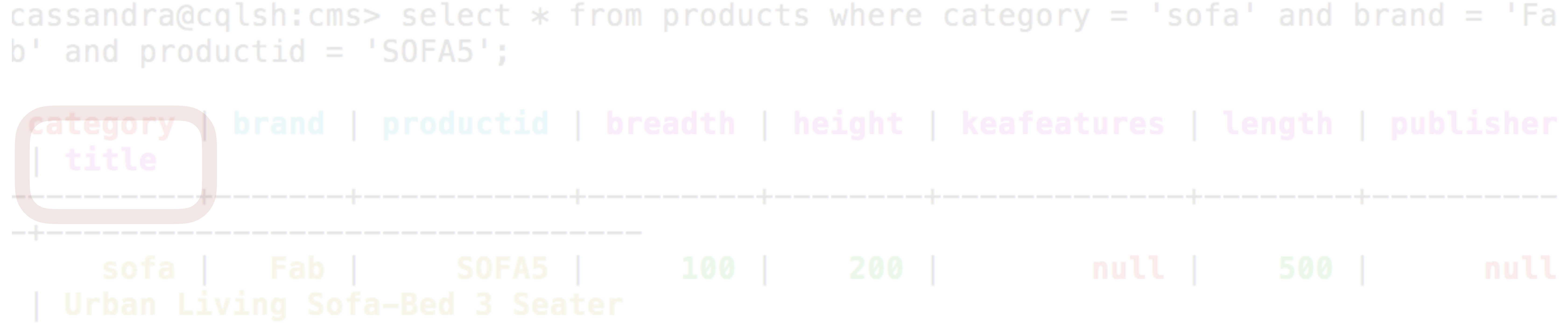
```
cassandra@cqlsh:cms> select * from products where category = 'sofa' and brand = 'Fa
b' and productid = 'SOFA5';

 category | brand | productid | breadth | height | keafeatures | length | publisher
 | title
----------+-------+-----------+---------+--------+-------------+--------+-----------
-+-----------------------------
     sofa |   Fab |     SOFA5 |     100 |    200 |        null |    500 |      null
 | Urban Living Sofa-Bed 3 Seater
```

Let's see the data of a listing and a product



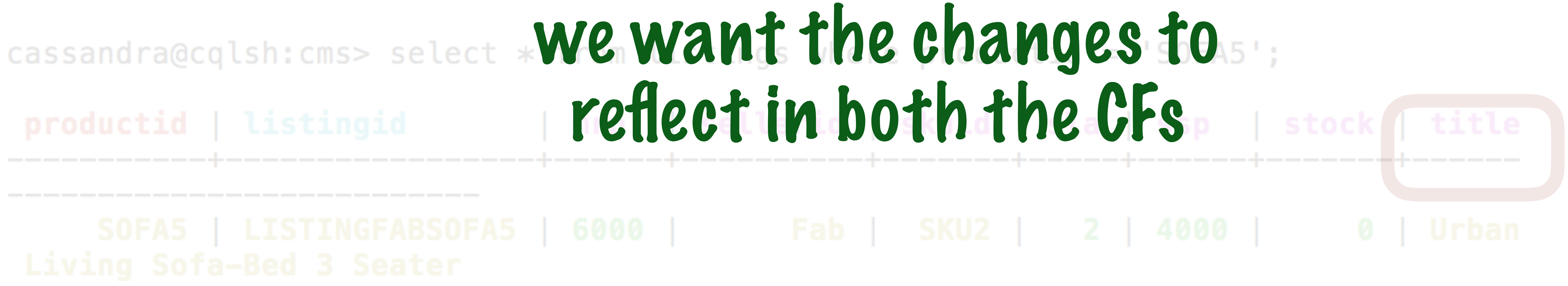
```
cassandra@cqlsh:cms> select * from listings where productid = 'SOFA5';

 productid | listingid          | mrp   | sellerid | skuid | sla | ssp   | stock | title
-----------+--------------------+-------+----------+-------+-----+-------+-------+------
-------------------------
     SOFA5 | LISTINGF[illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible]0 | Urban
 Living Sofa-Bed 3 Seater
```

If we want to update title, we want the changes to reflect in both the CFs



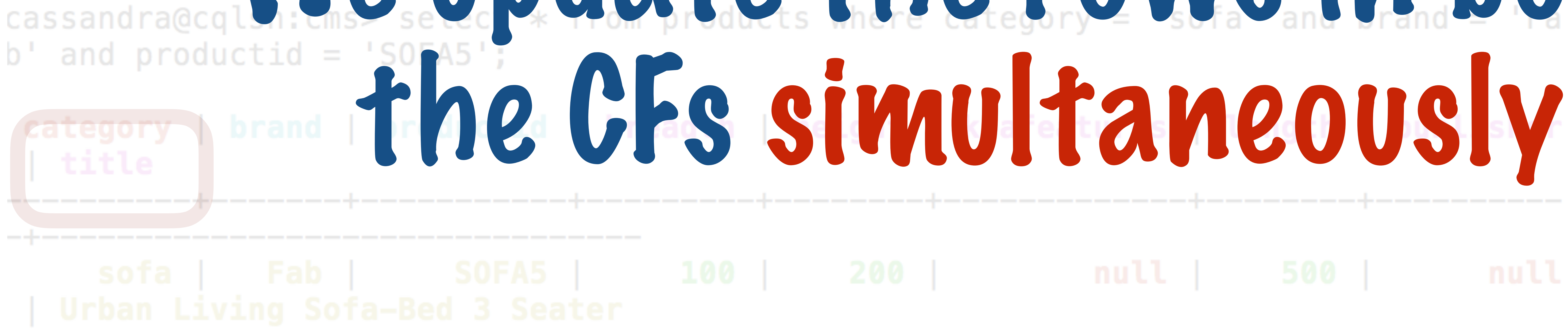
```
cassandra@cqlsh:cms> select * from products where category = 'sofa' and brand = 'Fa
b' and productid = 'SOFA5';

 category | brand | productid | breadth | height | keafeatures | length | publisher
 | title
----------+-------+-----------+---------+--------+-------------+--------+----------
-+-------------------------------
     sofa |   Fab |     SOFA5 |     100 |    200 |        null |    500 |      null
 | Urban Living Sofa-Bed 3 Seater
```

Let's see the data of a listing and a product



```
cassandra@cqlsh:cms> select * [illegible] gs whe[illegible] 'SOFA5';

 productid | listingid         | [illegible] | [illegible] | [illegible] | [illegible] | [illegible]sp | stock | title
-----------+-------------------+------+------------+---------+------+------+-------+-------
--------------------------
     SOFA5 | LISTINGFABSOFA5 | 6000 |        Fab |    SKU2 |    2 | 4000 |     0 | Urban
 Living Sofa-Bed 3 Seater
```

If we want to update title, we want the changes to reflect in both the CFs



```
cassandra@cqlsh:cms> selec[illegible] * from products where category = 'sofa' and brand = 'Fa
b' and productid = 'SOFA5';

 category | brand | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible]
 | title
----------+-------+-----------+-----------+----------+---------------+---------+---------
-+-----------------------------------
     sofa |   Fab |     SOFA5 |       100 |      200 |          null |     500 |     null
 | Urban Living Sofa-Bed 3 Seater
```

We update the rows in both the CFs simultaneously

We would need

simultaneous multiple updates

Either all statements are executed or none

logged BATCH does exactly that

LOGGED BATCH Statement

simultaneous multiple updates

Either all statements are executed or none

LOGGED BATCH Statement

simultaneous multiple updates

Either all statements are executed or none

Atomic **But not Isolated**

LOGGED BATCH Statement

Not Isolated

If the first statement is executed,
but the rest are still in process

Its changes are
visible to client

LOGGED BATCH Statement

To achieve atomicity

cassandra writes the batch
to the batch log system table

it maintains the entry till
the entire batch is executed

LOGGED BATCH Statement

To ensure its successful execution

2 replicas of batch log are created

if the coordinator node fails

another node which has the replica will take over

LOGGED BATCH Statement

This adds more load on the coordinator nodes and the cluster

Why should we use logged batch?

LOGGED BATCH Statement

Only to maintain consistency between denormalized tables

(which is our use case)

They are not meant to be used for normal updates

Let's change the title of SOFA5 in both product and listing using logged batch

Let's go through the java code

We add a new class
CommonPersistenceHandler

contains methods to perform db operations
involving both product and listing

it has the updateTitle() method
to update the title

We will use 2 classes here

CommonPersistenceHandler - it has updateTitle() method

Main class - to build the input parameters to call updateTitle()

Let's see the steps we will follow

1. build input data for updateTitle
2. prepare update statement for listings
3. prepare update statements for products
4. execute the batch

Let's see the steps we will follow

1. build input data for updateTitle
2. prepare update statement for listings
3. prepare update statements for products
4. execute the batch

Main Class

1. build input data for updateTitle

```
public class Main {
     public static void main(String[] args){
         updateTitle();

    }

   static void updateTitle() {
       CommonPersistenceHandler persistenceHandler = new CommonPersistenceHandler();

       Listing listing = new Listing();
       listing.setListingId("LISTINGFABSOFA5");
       Map<String, Object> attributes = Maps.newHashMap();
       attributes.put(AttributeNames.PRODUCTID.getValue(), "SOFA5");

       attributes.put(AttributeNames.TITLE.getValue(), "Urban Living Sofa-Bed 3 Seater");
       listing.setAttributes(attributes);


       Product product = new Product();
       product.setProductId("SOFA5");

       Map<String, Object> attributesMap = Maps.newHashMap();
       attributesMap.put(AttributeNames.CATEGORY.getValue(), "sofa");
       attributesMap.put(AttributeNames.BRAND.getValue(), "Fab");

       attributesMap.put(AttributeNames.TITLE.getValue(), "Urban Living Sofa-Bed 3 Seater");
       product.setAttributesMap(attributesMap);

       persistenceHandler.updatetitle(listing, product);
    }
}
```

Writes in cassandra are always done in append mode

To update data, we need to provide

primary key

the columns that we want to update

Main Class

1. build input data for updateTitle

We want to update only title for this listing

```
public class Main {
    public static void main(String[] args) {
        updateTitle();

    }

    static void updateTitle() {
        CommonPersistenceHandler persistenceHandler = new CommonPersistenceHandler();

        Listing listing = new Listing();
        listing.setListingId("LISTINGFABSOFA5");
        Map<String, Object> attributes = Maps.newHashMap();
        attributes.put(AttributeNames.PRODUCTID.getValue(), "SOFA5");

        attributes.put(AttributeNames.TITLE.getValue(), "Urban Living Sofa-Bed 3 Seater");
        listing.setAttributes(attributes);

        Product product = new Product();
        product.setProductId("SOFA5");

        Map<String, Object> attributesMap = Maps.newHashMap();
        attributesMap.put(AttributeNames.CATEGORY.getValue(), "sofa");
        attributesMap.put(AttributeNames.BRAND.getValue(), "Fab");

        attributesMap.put(AttributeNames.TITLE.getValue(), "Urban Living Sofa-Bed 3 Seater");
        product.setAttributesMap(attributesMap);

        persistenceHandler.updatetitle(listing, product);
    }
}
```

We will create a Listing object with primary key data productid, listingid title data

Main Class

1. build input data for updateTitle

```
public class Main {
    public static void main(String[] args){
        updateTitle();

   }

  static void updateTitle() {
     CommonPersistenceHandler persistenceHandler = new CommonPersistenceHandler();

     Listing listing = new Listing();
     listing.setListingId("LISTINGFABSOFA5");
     Map<String, Object> attributes = Maps.newHashMap();
     attributes.put(AttributeNames.PRODUCTID.getValue(), "SOFA5");

     attributes.put(AttributeNames.TITLE.getValue(), "Urban Living Sofa-Bed 3 Seater1");
     listing.setAttributes(attributes);

     Product product = new Product();
      product.setProductId("SOFA5");

      Map<String, Object> attributesMap = Maps.newHashMap();
      attributesMap.put(AttributeNames.CATEGORY.getValue(), "sofa");
      attributesMap.put(AttributeNames.BRAND.getValue(), "Fab");

      attributesMap.put(AttributeNames.TITLE.getValue(), "Urban Living Sofa-Bed 3 Seater");
      product.setAttributesMap(attributesMap);

      persistenceHandler.updatetitle(listing, product);


    }
}
```

Similarly for product

We will create a Product object with primary key data category, brand, productid title data

Let's see the steps we will follow

1. build input data for updateTitle
2. prepare update statement for listings
3. prepare update statements for products
4. execute the batch

CommonPersistenceHandler

2. prepare update statement for listings

```
public class CommonPersistenceHandler {

    private static String keyspace = "cms";
    private static String listingColumnFamily = "listings";
    private static String productColumnFamily = "products";

    public void updatetitle(Listing listing, Product product){

      BatchStatement batch = new BatchStatement();
      if(listing != null) {
          batch.addAll(getListingUpdateStatements(listing));
      }
      if(product != null) {
          batch.addAll(getProductUpdateStatements(product));
      }

      try {
          Session session = Connector.getSession();
          session.execute(batch);
      }catch (Exception e){
          e.printStackTrace();
      }
}

private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
       for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
```

In updateTitle(),

We create a BatchStatement object

By default it is logged batch

CommonPersistenceHandler

2. prepare update statement for listings

```
public class CommonPersistenceHandler {

    private static String keyspace = "cms";
    private static String listingColumnFamily = "listings";
    private static String productColumnFamily = "products";

    public void updatetitle(Listing listing, Product product){

      BatchStatement batch = new BatchStatement();
      if(listing != null) {
          batch.addAll(getListingUpdateStatements(listing));
      }
      if(product != null) {
          batch.addAll(getProductUpdateStatements(product));
      }

      try {
          Session session = Connector.getSession();
          session.execute(batch);
      }catch (Exception e){
          e.printStackTrace();
      }
}

private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
       for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
```

next we add listing and product update statements

getListingUpdateStatements() adds statements for listing CF

getProductUpdateStatements() adds statements for products CF

CommonPersistenceHandler

2. prepare update statement for listings

```
    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null) {
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

This returns multiple update statements

```
private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
       for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                    .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                    .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                    .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
            updates.add(updateStatement);
        }
    }

    return updates;
}
```

one for each column we update

```
    private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName));
    }

  }
```

CommonPersistenceHandler

2. prepare update statement for listings

We iterate over the attributes provide in Listing object

```
    public void updateTitle(Listing listing, Product product){

    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null)
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}

private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
       for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                    .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                    .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                    .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
            updates.add(updateStatement);
        }
    }

    return updates;
}


   private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName));
    }

  }
```

create a separate update statement for each of them

and add them to a list of updates for batch

CommonPersistenceHandler

2. prepare update statement for listings

```
    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null) {
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connection.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

attributeset contains Listing data

```
private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
     for(String attributeName : attributeSet.keySet()){
      if(!isPartOfPrimaryKeyForListing(attributeName)) {

          Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                  .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                  .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                  .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
          updates.add(updateStatement);
      }
    }

    return updates;
}

private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName));
    }

  }
```

CommonPersistenceHandler

2. prepare update statement for listings

```
    public void updateTitle(Listing listing, Product product){

        BatchStatement batch = new BatchStatement();
        if(listing != null) {
            batch.addAll(getListingUpdateStatements(listing));
        }
        if(product != null) {
            batch.addAll(getProductUpdateStatements(product));
        }

        try {
            Session session = connector.getSession();
            session.execute(batch);
        }catch (Exception e){
            e.printStackTrace();
        }
}
```

The primary key cannot be updated

```
private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
        for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                    .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                    .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                    .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
            updates.add(updateStatement);
        }
    }

    return updates;
}
private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName));
    }
  }
```

so that updateStatements are not build for them attribute

CommonPersistenceHandler

2. prepare update statement for listings

```
    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null) {
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

update cms.listings

```
private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
       for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                    .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                    .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                    .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
                updates.add(updateStatement);
            }
    }

    return updates;
}

private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName));
    }
}
```

We will use the update() method of QueryBuilder

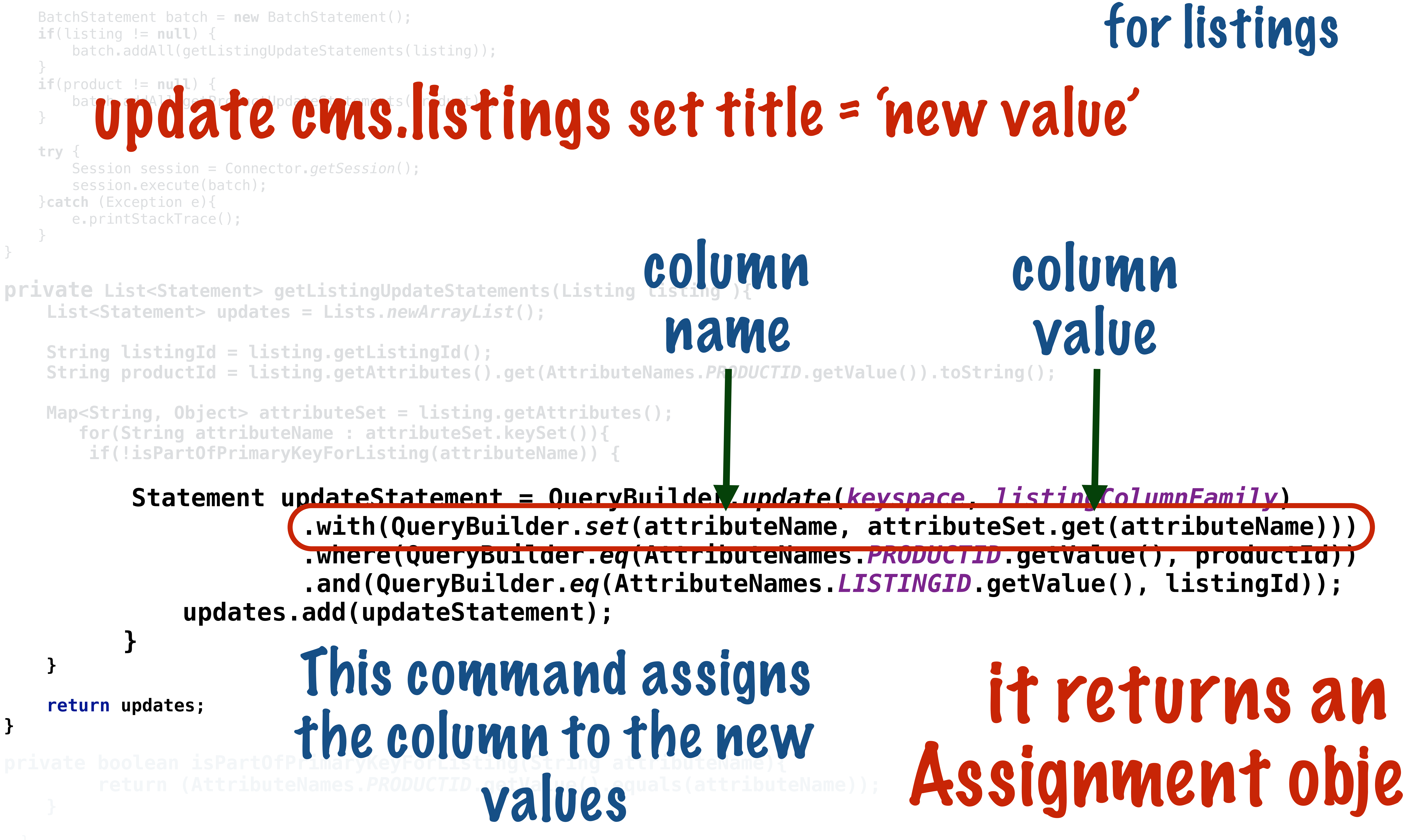
update() starts the update query

it returns in-building Update object

CommonPersistenceHandler

2. prepare update statement for listings

update cms.listings set title = ‘new value’



```java
    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null) {
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}

private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
        for(String attributeName : attributeSet.keySet()){
         if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                            .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                            .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                            .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
                updates.add(updateStatement);
            }
    }

    return updates;
}

private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName));
    }
```

column name

column value

This command assigns the column to the new values

it returns an Assignment object

CommonPersistenceHandler

2. prepare update statement for listings

```
    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null) {
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

update cms.listings set title = ‘new value’
where productid = ‘ id’

```
private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
       for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                            .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                            .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                            .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
                updates.add(updateStatement);
            }
    }

    return updates;
}

private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName));
    }
```

first we set a condition on the partition key productid

where() adds a where clause for the assignments

it returns in-building Update.Where clause object

CommonPersistenceHandler

2. prepare update statement for listings

```
    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null) {
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

update cms.listings set title = 'new value'
where productid = ' id'

```
private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getVal[illegible]

    Map<String, Object> attributeSet = listing.getAttributes();
       for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                      .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                      .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                      .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
              updates.add(updateStatement);
          }
    }

    return updates;
}

private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName) [illegible]
    }
```

the update statement can be be executed now

set and where are required for a valid UpdateStatement

CommonPersistenceHandler

2. prepare update statement for listings

update cms.listings set title = 'new value' where productid = ' id' and listingid = ' id'

```
    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null) {
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}

private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
       for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForListing(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                        .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                        .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                        .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
                updates.add(updateStatement);
        }
    }

    return updates;
}

private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName))
    }
```

now a condition on the clustering key listingid

and() adds a where clause is added for the assignments

it returns in-building Update.Where object

CommonPersistenceHandler

2. prepare update statement for listings

update cms.listings set title = 'new value'
where productid = ' id' and listingid = ' id'

```
    BatchStatement batch = new BatchStatement();
    if(listing != null) {
        batch.addAll(getListingUpdateStatements(listing));
    }
    if(product != null) {
        batch.addAll(getProductUpdateStatements(product));
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}

private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
        for(String attributeName : attributeSet.keySet()){
          if(!isPartOfPrimaryKeyForListing(attributeName)) {

             Statement updateStatement = QueryBuilder.update(keyspace, listingColumnFamily)
                         .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                         .where(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId))
                         .and(QueryBuilder.eq(AttributeNames.LISTINGID.getValue(), listingId));
                updates.add(updateStatement);
          }
    }

    return updates;
}
private boolean isPartOfPrimaryKeyForListing(String attributeName){
        return (AttributeNames.PRODUCTID.getValue().equals(attributeName));
    }

    }
```

we add the update statement to the list

Let's see the steps we will follow

1. build input data for updateTitle
2. prepare update statement for listings
3. **prepare update statements for products**
4. execute the batch

CommonPersistenceHandler

3. prepare update statements for products

```java
        }

        try {
            Session session = Connector.getSession();
            session.execute(batch);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
    private List<Statement> getProductUpdateStatements(Product product){
        List<Statement> updates = Lists.newArrayList();

        String productId = product.getProductId();
        String category = product.getAttributesMap().get(AttributeNames.CATEGORY.getValue()).toString();
        String brand = product.getAttributesMap().get(AttributeNames.BRAND.getValue()).toString();

        Map<String, Object> attributeSet = product.getAttributesMap();
        // every attribute update is a separate statement
        for(String attributeName : attributeSet.keySet()){
            if(!isPartOfPrimaryKeyForProduct(attributeName)) {
                Statement updateStatement = QueryBuilder.update(keyspace, productColumnFamily)
                        .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                        .where(QueryBuilder.eq(AttributeNames.CATEGORY.getValue(), category))
                        .and(QueryBuilder.eq(AttributeNames.BRAND.getValue(), brand))
                        .and(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId));

                updates.add(updateStatement);
            }
        }
        return updates;
    }

    private boolean isPartOfPrimaryKeyForProduct(String attributeName){
        return (AttributeNames.CATEGORY.getValue().equals(attributeName) ||
                AttributeNames.BRAND.getValue().equals(attributeName));
    }
```

Like listings, here also we iterate over all the attributes and create a separate update statement for them

excluding primary key attributes - category, brand

CommonPersistenceHandler

3. prepare update statements for products

```java
        }

        try {
            Session session = Connector.getSession();
            session.execute(batch);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
    private List<Statement> getProductUpdateStatements(Product product){
        List<Statement> updates = Lists.newArrayList();

        String productId = product.getProductId();
        String category = product.getAttributesMap().get(AttributeNames.CATEGORY.getValue()).toString();
        String brand = product.getAttributesMap().get(AttributeNames.BRAND.getValue()).toString();

        Map<String, Object> attributeSet = product.getAttributesMap();
        // every attribute update is a separate statement
        for(String attributeName : attributeSet.keySet()){
            if(!isPartOfPrimaryKeyForProduct(attributeName)) {

                Statement updateStatement = QueryBuilder.update(keyspace, productColumnFamily)
                        .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                        .where(QueryBuilder.eq(AttributeNames.CATEGORY.getValue(), category))
                        .and(QueryBuilder.eq(AttributeNames.BRAND.getValue(), brand))
                        .and(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId));

                updates.add(updateStatement);
            }
        }
        return updates;
    }

    private boolean isPartOfPrimaryKeyForProduct(String attributeName){
        return (AttributeNames.CATEGORY.getValue().equals(attributeName) ||
                AttributeNames.BRAND.getValue().equals(attributeName));
    }
```

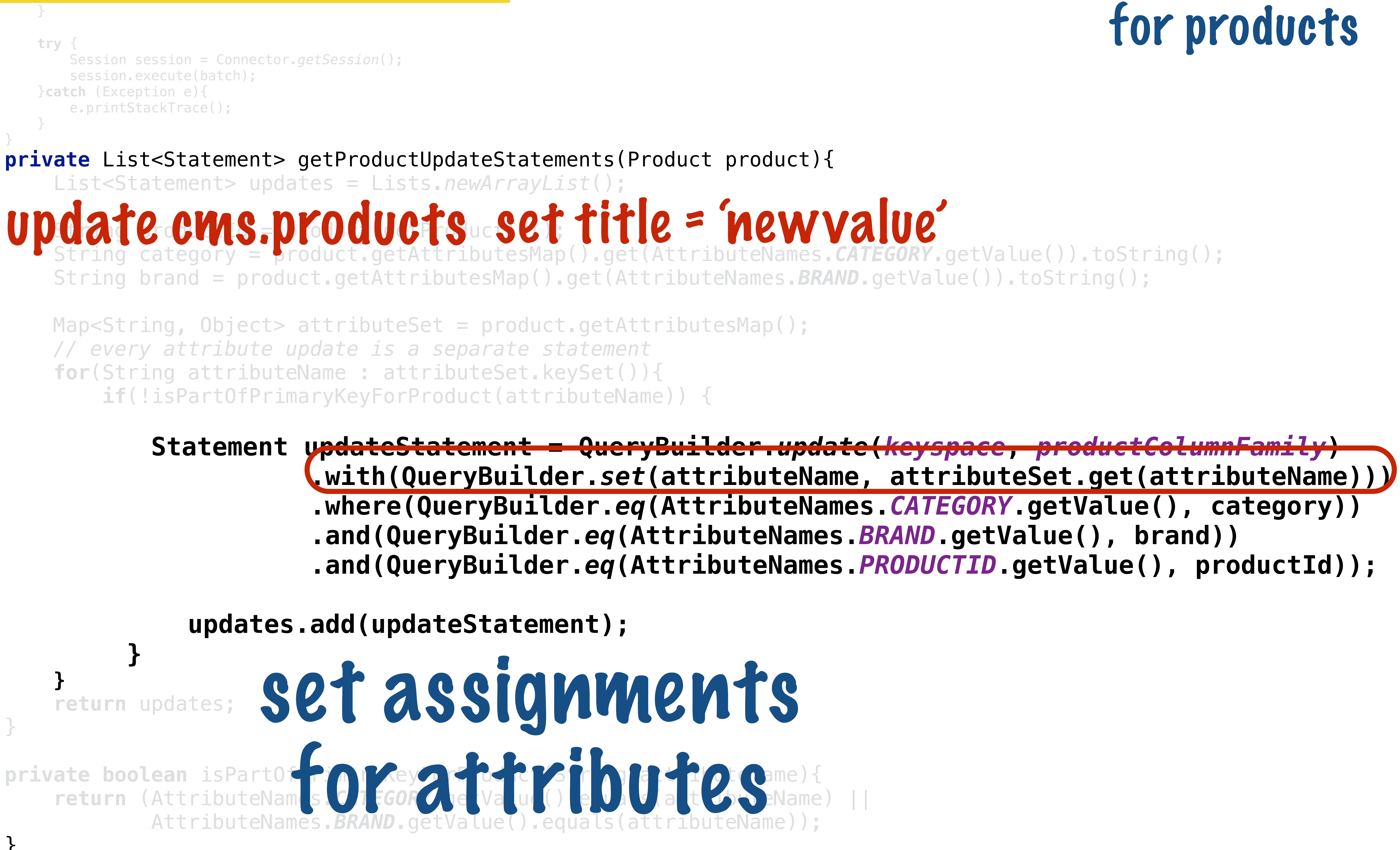
update cms.products

we start building the update statement with update()

with parameters keyspace=cms and CF as products

CommonPersistenceHandler

3. prepare update statements for products



```
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}
private List<Statement> getProductUpdateStatements(Product product){
    List<Statement> updates = Lists.newArrayList();

    String productId = product.getProductId();
    String category = product.getAttributesMap().get(AttributeNames.CATEGORY.getValue()).toString();
    String brand = product.getAttributesMap().get(AttributeNames.BRAND.getValue()).toString();

    Map<String, Object> attributeSet = product.getAttributesMap();
    // every attribute update is a separate statement
    for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForProduct(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, productColumnFamily)
                    .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                    .where(QueryBuilder.eq(AttributeNames.CATEGORY.getValue(), category))
                    .and(QueryBuilder.eq(AttributeNames.BRAND.getValue(), brand))
                    .and(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId));

            updates.add(updateStatement);
        }
    }
    return updates;
}

private boolean isPartOfPrimaryKeyForProduct(String attributeName){
    return (AttributeNames.CATEGORY.getValue().equals(attributeName) ||
            AttributeNames.BRAND.getValue().equals(attributeName));
}
```

update cms.products set title = 'newvalue'

set assignments for attributes

CommonPersistenceHandler

3. prepare update statements for products

```
        }

        try {
            Session session = Connector.getSession();
            session.execute(batch);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
    private List<Statement> getProductUpdateStatements(Product product){
        List<Statement> updates = Lists.newArrayList();

        String productId = product.getProductId();
        String category = product.getAttributesMap().get(AttributeNames.CATEGORY.getValue()).toString();
        String brand = product.getAttributesMap().get(AttributeNames.BRAND.getValue()).toString();

        Map<String, Object> attributeSet = product.getAttributesMap();
        // every attribute update is a separate statement
        for(String attributeName : attributeSet.keySet()){
            if(!isPartOfPrimaryKeyForProduct(attributeName)) {

                Statement updateStatement = QueryBuilder.update(keyspace, productColumnFamily)
                        .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                        .where(QueryBuilder.eq(AttributeNames.CATEGORY.getValue(), category))
                        .and(QueryBuilder.eq(AttributeNames.BRAND.getValue(), brand))
                        .and(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId));

                updates.add(updateStatement);
            }
        }
        return updates;
    }

    private boolean isPartOfPrimaryKeyForProduct(String attributeName){
        return (AttributeNames.CATEGORY.getValue().equals(attributeName) ||
                AttributeNames.BRAND.getValue().equals(attributeName));
    }
```

update cms.products set title = ‘newvalue’ where category = ‘value’

add where clause with first condition on partition key category

CommonPersistenceHandler

3. prepare update statements for products

```
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}
private List<Statement> getProductUpdateStatements(Product product){
    List<Statement> updates = Lists.newArrayList();

    String productId = product.getProductId();
    String category = product.getAttributesMap().get(AttributeNames.CATEGORY.getValue()).toString();
    String brand = product.getAttributesMap().get(AttributeNames.BRAND.getValue()).toString();

    Map<String, Object> attributeSet = product.getAttributesMap();
    // every attribute update is a separate statement
    for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForProduct(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, productColumnFamily)
                    .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                    .where(QueryBuilder.eq(AttributeNames.CATEGORY.getValue(), category))
                    .and(QueryBuilder.eq(AttributeNames.BRAND.getValue(), brand))
                    .and(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId));

            updates.add(updateStatement);
        }
    }
    return updates;
}

private boolean isPartOfPrimaryKeyForProduct(String attributeName){
    return (AttributeNames.CATEGORY.getValue().equals(attributeName) ||
            AttributeNames.BRAND.getValue().equals(attributeName));
}
```

update cms.products set title = 'newvalue' where category = 'value' and brand = 'value'

next condition on 1st clustering key brand

CommonPersistenceHandler

3. prepare update statements for products

```
    }

    try {
        Session session = Connector.getSession();
        session.execute(batch);
    }catch (Exception e){
        e.printStackTrace();
    }
}
private List<Statement> getProductUpdateStatements(Product product){
    List<Statement> updates = Lists.newArrayList();

    String productId = product.getProductId();
    String category = product.getAttributesMap().get(AttributeNames.CATEGORY.getValue()).toString();
    String brand = product.getAttributesMap().get(AttributeNames.BRAND.getValue()).toString();

    Map<String, Object> attributeSet = product.getAttributesMap();
    // every attribute update is a separate statement
    for(String attributeName : attributeSet.keySet()){
        if(!isPartOfPrimaryKeyForProduct(attributeName)) {

            Statement updateStatement = QueryBuilder.update(keyspace, productColumnFamily)
                    .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                    .where(QueryBuilder.eq(AttributeNames.CATEGORY.getValue(), category))
                    .and(QueryBuilder.eq(AttributeNames.BRAND.getValue(), brand))
                    .and(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId));

            updates.add(updateStatement);
        }
    }
    return updates;
}

private boolean isPartOfPrimaryKeyForProduct(String attributeName){
    return (AttributeNames.CATEGORY.getValue().equals(attributeName) ||
            AttributeNames.BRAND.getValue().equals(attributeName));
}
```

update cms.products set title = 'newvalue' where category = 'value' and brand = 'value' and productid = 'id';

and then on productid

CommonPersistenceHandler

3. prepare update statements for products

```
        }

        try {
            Session session = Connector.getSession();
            session.execute(batch);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
    private List<Statement> getProductUpdateStatements(Product product){
        List<Statement> updates = Lists.newArrayList();

        String productId = product.getProductId();
        String category = product.getAttributesMap().get(AttributeNames.CATEGORY.getValue()).toString();
        String brand = product.getAttributesMap().get(AttributeNames.BRAND.getValue()).toString();

        Map<String, Object> attributeSet = product.getAttributesMap();
        // every attribute update is a separate statement
        for(String attributeName : attributeSet.keySet()){
            if(!isPartOfPrimaryKeyForProduct(attributeName)) {

                Statement updateStatement = QueryBuilder.update(keyspace, productColumnFamily)
                                .with(QueryBuilder.set(attributeName, attributeSet.get(attributeName)))
                                .where(QueryBuilder.eq(AttributeNames.CATEGORY.getValue(), category))
                                .and(QueryBuilder.eq(AttributeNames.BRAND.getValue(), brand))
                                .and(QueryBuilder.eq(AttributeNames.PRODUCTID.getValue(), productId));

                updates.add(updateStatement);
            }
        }
        return updates;
    }

    private boolean isPartOfPrimaryKeyForProduct(String attributeName){
        return (AttributeNames.CATEGORY.getValue().equals(attributeName) ||
                AttributeNames.BRAND.getValue().equals(attributeName));
    }
```

update cms.products set title = 'newvalue' where category = 'value'
and brand = 'value' and productid = 'id';

add this update to the list for batch

Let's see the steps we will follow

1. build input data for updateTitle
2. prepare update statement for listings
3. prepare update statements for products
4. execute the batch

CommonPersistenceHandler

4. execute the batch

```
public class CommonPersistenceHandler {

    private static String keyspace = "cms";
    private static String listingColumnFamily = "listings";
    private static String productColumnFamily = "products";

    public void updatetitle(Listing listing, Product product){

        BatchStatement batch = new BatchStatement();
        if(listing != null) {
            batch.addAll(getListingUpdateStatements(listing));
        }
        if(product != null) {
            batch.addAll(getProductUpdateStatements(product));
        }

        try {
            Session session = Connector.getSession();
            session.execute(batch);
        }catch (Exception e){
            e.printStackTrace();
        }
    }

    private List<Statement> getListingUpdateStatements(Listing listing){
        List<Statement> updates = Lists.newArrayList();

        String listingId = listing.getListingId();
        String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

        Map<String, Object> attributeSet = listing.getAttributes();
        for (String attributeName : attributeSet.keySet()){
```

we have added the update statements to the batch

let's now execute the batch

CommonPersistenceHandler

4. execute the batch

```java
public class CommonPersistenceHandler {

    private static String keyspace = "cms";
    private static String listingColumnFamily = "listings";
    private static String productColumnFamily = "products";

    public void updatetitle(Listing listing, Product product){

        BatchStatement batch = new BatchStatement();
        if(listing != null) {
            batch.addAll(getListingUpdateStatements(listing));
        }
        if(product != null) {
            batch.addAll(getProductUpdateStatements(product));
        }

        try {
            Session session = Connector.getSession();
            session.execute(batch);
        }catch (Exception e){
            e.printStackTrace();
        }
}

private List<Statement> getListingUpdateStatements(Listing listing ){
    List<Statement> updates = Lists.newArrayList();

    String listingId = listing.getListingId();
    String productId = listing.getAttributes().get(AttributeNames.PRODUCTID.getValue()).toString();

    Map<String, Object> attributeSet = listing.getAttributes();
```

we get the session object from Connector

and execute the batch