Baze de date nerelationale

Dumitrescu Delia Ioana

December 2020

1 Introducere

Bazele de date nerelationale permit manipularea unor date nestructurate, fiind flexibile atat din punct de vedere al datelor stocate, cat si al scalabilitatii. Acestea renunta la o parte din constrangerile unui model relational cu scopul de a imbunatati performanta. Astfel, modelul *NoSQL* nu satisface in intregime proprietatile unei baze de date *SQL*, anume *Atomicitate*(nu exista posibilitatea completarii unei tranzactii in mod incomplet, fie este completata si validata, fie abandonata), *Consistenta*(efectuarea unor modificari corecte asupra bazei de date, adica pastrarea unei stari consistente de la o schimbare la alta), *Izolare*(schimbarile efectuate asupra bazei de date sunt vizibile abia dupa validarea lor) si *Durabilitate*(modificarile asupra bazei de date nu sunt pierdute) **(ACID)**. Totusi, bazele de date nerelationale sunt usor adaptabile la schimbari si suporta cantitati foarte mari de date.

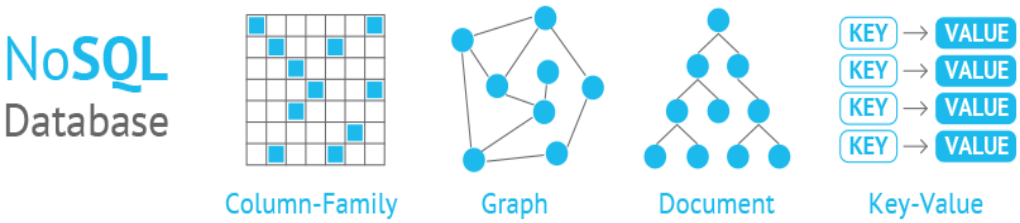
2 Diferente intre baze de date relationale(*SQL*) si baze de date nerelationale(*NoSQL*)

- **Modul de stocare al datelor**:
  - *SQL*: La nivel de tabel
  - *NoSQL*: Diferite moduri de stocare(prezentate mai jos)
- **Nivel de flexibilitate**:
  - *SQL*: Rigide, necesita o schema predefinita
  - *NoSQL*: Foarte flexibile, suporta o schema dinamica, care se schimba in mod constant
- **Scalabilitate**:
  - *SQL*: Scalate in mod vertical, in cazul in care este nevoie de mai multa putere pentru stocare, este necesara imbunatatirea RAM-ului, a SSD-ului sau a CPU-ului(extra hardware). Nu se poate adauga alt server.

- *NoSQL*: Scalate in mod orizontal, in cazul in care este nevoie de mai multa putere pentru stocare, se pot adauga pur si simplu mai multe servere

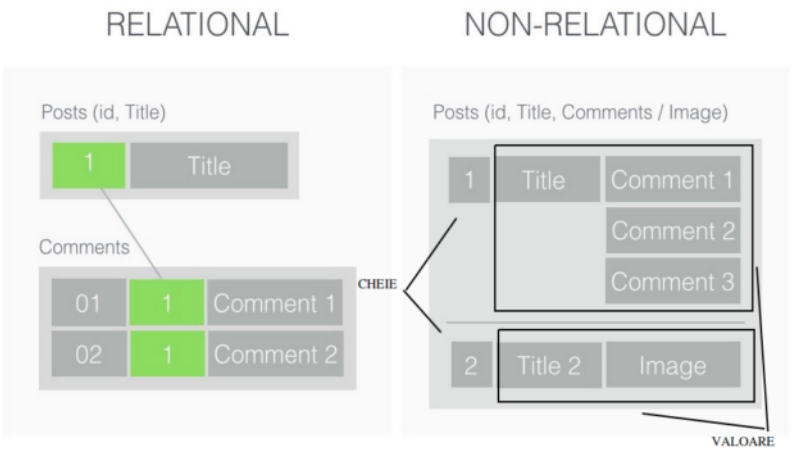
- **Cand sa fie folosite**:
  - *SQL*: Ideale intr-un mediu cu query-uri complexe si intensive, in care corectitudinea datelor este esentiala. Bazele de date *SQL* sunt mai stabile si asigura integritatea datelor.
  - *NoSQL*: Ideale atunci cand cerintele se schimba in mod constant, se doreste extragerea datelor foarte rapid, iar integritatea si consistenta datelor nu sunt esentiale

3 Modul de stocare al datelor

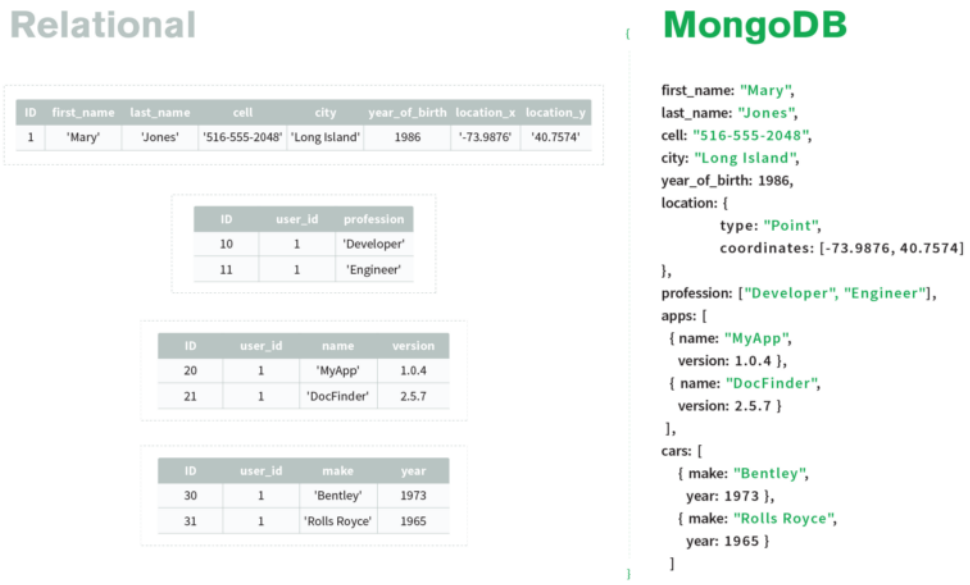


In functie de modul de stocare al datelor, exista mai multe tipuri de baze de date *NoSQL*:

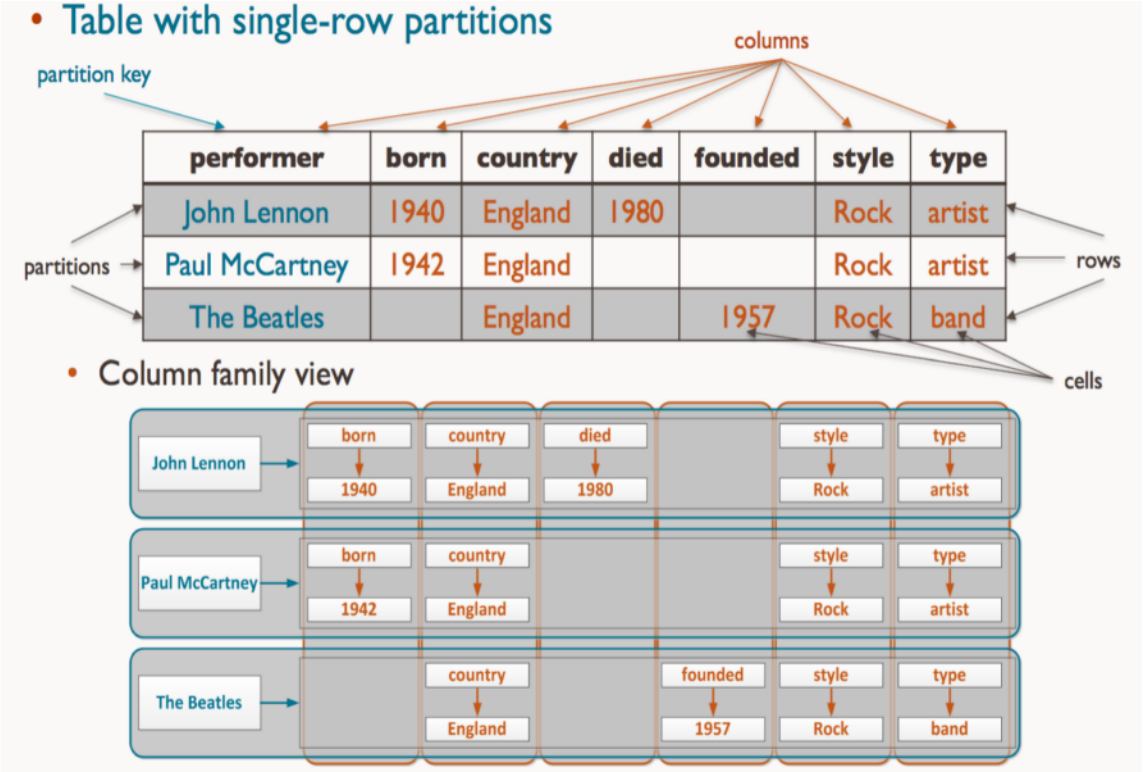
- De tip **cheie-valoare** : asociaza o cheie(un mod prin care se vor accesa datele) unei valori(care poate reprezenta orice, de la un simplu *string* la date mai complexe). Este, in linii mari, o varianta mai avansata a dictionarelor din limbajele de programare clasice. Acest mod de stocare a datelor este util atunci cand se doreste stocare unei cantitati mari de date si nu este nevoie de query-uri complexe pentru a le accesa.



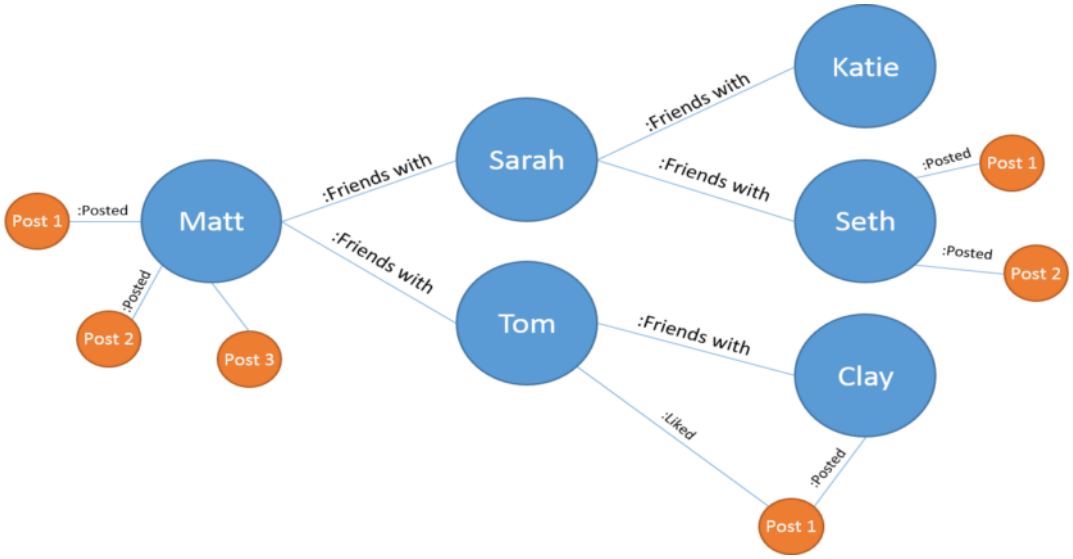
- De tip **document** : reprezinta o specializare a bazelor de date de tip cheie-valoare, deoarece fiecare document din baza de date corespunde unei chei. Documentele encapsuleaza datele sub diferite forme: JSON, BSON, BLOB, XML si altele.



- De tip **familie de coloane**: spre deosebire de bazele de date de tip *SQL*, datele sunt stocate la nivel de coloana, nu de linie. Acest mod de stocare este util deoarece doua randuri nu trebuie sa contina aceleasi coloane. De asemenea, este posibila accesarea directa a unei coloane precum "born" din exemplul de mai jos, fara a fi necesara o iteratie a fiecarui "performer".



- De tip **graf** : stocheaza datele in noduri (datele pe care vrem sa le retinem) si muchii(relatiile dintre noduri).



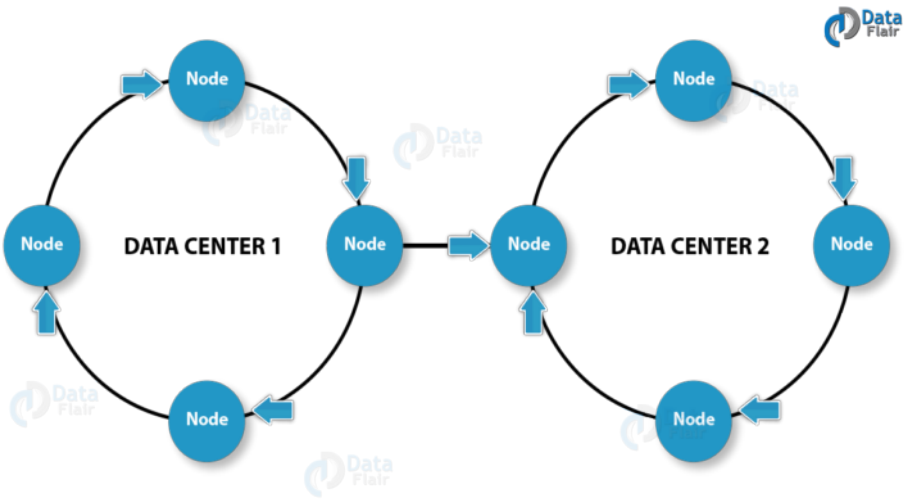
- De tip **multi-model**: inglobeaza mai multe modele de date

4 Exemplu realizat folosind baza de date Apache Cassandra

Apache Cassandra este o baza de date open source utilizata pentru gestionarea unor cantitati foarte mari de date structurate raspandite in toata lumea. Aceasta ofera atat rezultate foarte bune pentru scalarea performantei, cat si si posibilitatea de a distribui in mod partitionat setul de date pe nodurile din cluster.

O baza de date de tip Apache Cassandra este formata din mai multe *Data Centers*, care formeaza un *Cluster*. Fiecare *Data Center* e alcatuit din mai multe *noduri*, unde este stocata informatia. Datele sunt stocate in mai multe zone dintr-un *inel*, astfel ca, in cazul unei defectiuni, datele nu sunt pierdute(exista replici pentru ele).

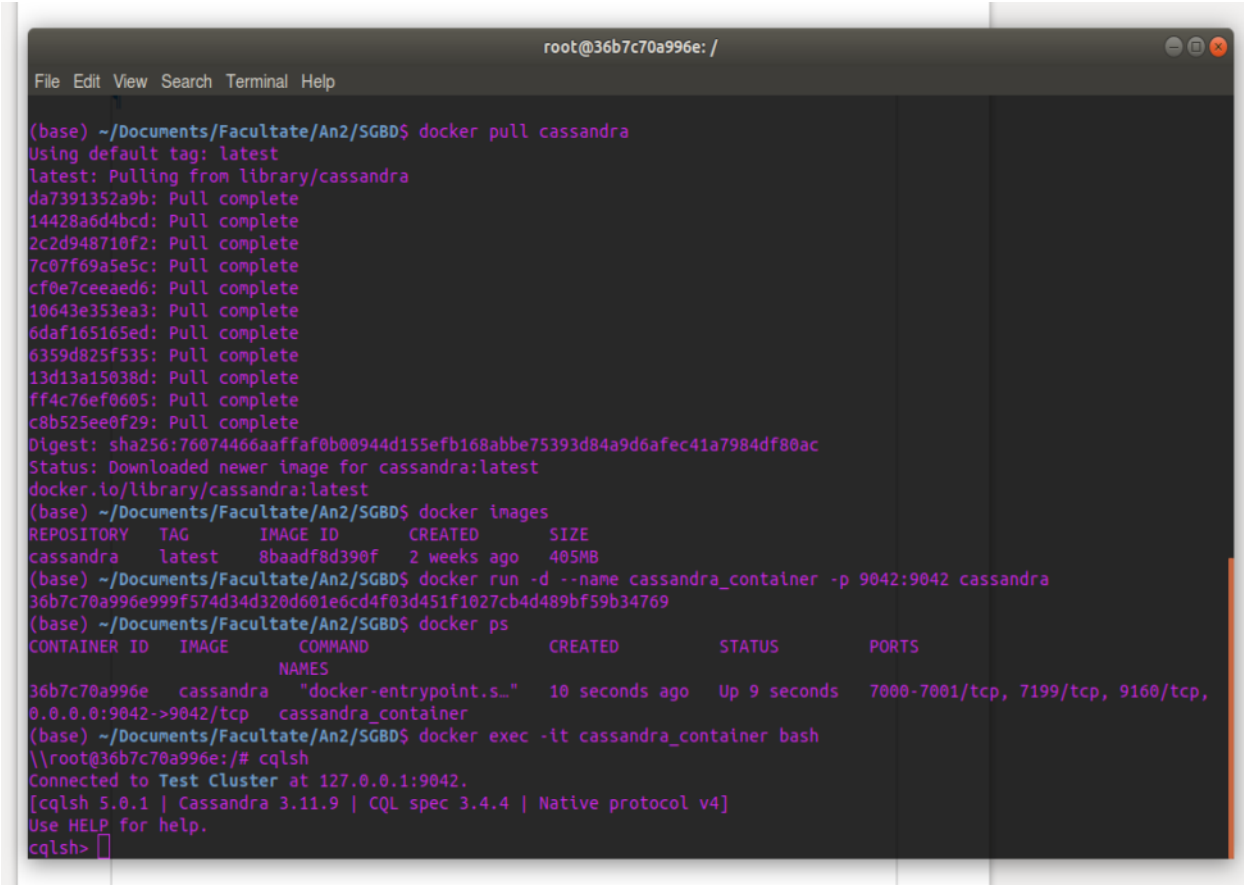
*A*pache Cassandra retine datele in familii de coloane. Keyspace-ul este un container pentru o lista cu una sau mai multe familii de coloane. O familie de coloane este un container al unei colectii de randuri. Fiecare rand contine coloane ordonate. Fiecare *keyspace* contine minim o familie de coloane.



4.1 Configurarea mediului de lucru

Pentru a putea rula comezi in *CQLSH*, vom folosi *Docker*. *CQLSH* este un shell pentru command line folosit pentru interactiunea cu Cassandra prin *CQL*, adica *Cassandra Query Language* . *Docker* este o platforma software ce permite construirea de aplicatii ce ruleaza intr-un container. Aceste containere sunt niste medii de executie izolate unele de celelalte, dar care impart nucleul de baza al sistemului de operare. Astfel, pachete instalate in diferite programe nu interfereaza si nu genereaza probleme. Ele sunt mult mai putin costisitoare ca masinile virtuale.

- Pasul 1: Instalare Docker: se urmeaza pasii de la https://docs.docker.com/engine/install/ubuntu/
- Pasul 2: Folosind Docker, configuram un container pornind de la o imagine de Cassandra si intram in acesta, unde urmeaza sa executam comenzi *CQL*.



4.2 Crearea unui KEYSPACE si includerea lui in mediul de lucru

Vom crea un *keyspace* pentru o activitate comerciala. Vrem sa obtinem informatii despre produse, cumparatori, despre toti cumparatorii carora le place un anumit produs si despre toate produsele pe care un cumparator le apreciaza.

4.3 Crearea tabelelor

```
CREATE TABLE Customer (cust_id text,
                       first_name text,
                       last_name text,
                       registered_on timestamp,
```

```
                        PRIMARY KEY (cust_id));

CREATE TABLE Product (prod_id text,
                      title text,
                      PRIMARY KEY (prod_id));

CREATE TABLE Product_Liked_By_Customer (cust_id text,
                                        first_name text,
                                        last_name text,
                                        liked_prod_id text,
                                        liked_on timestamp,
                                        title text,
                                        PRIMARY KEY (cust_id, liked_on));

CREATE TABLE Customer_By_Liked_Product (liked_prod_id text,
                                        liked_on timestamp,
                                        title text,
                                        cust_id text,
                                        first_name text,
                                        last_name text,
                                        PRIMARY KEY (liked_prod_id, liked_on));
```

4.4 Popularea tabelelor cu date

Tabelul Customer:

```
INSERT INTO commerce_data.Customer (cust_id, first_name, last_name,
registered_on) VALUES ('1', 'Delia', 'Dumitrescu', '2017-05-05');

INSERT INTO commerce_data.Customer (cust_id, first_name, last_name,
registered_on) VALUES ('2', 'Gabriel', 'Popescu', '2019-07-05');

INSERT INTO commerce_data.Customer (cust_id, first_name, last_name,
registered_on) VALUES ('3', 'Adrian', 'Munteanu', '2020-01-01');

INSERT INTO commerce_data.Customer (cust_id, first_name, last_name,
registered_on) VALUES ('4', 'Sorina', 'Alexe', '2009-07-09');
```

Tabelul Product:

```
INSERT INTO commerce_data.Product (prod_id, title) VALUES ('1', 'Cana');

INSERT INTO commerce_data.Product (prod_id, title) VALUES ('2', 'Prosop');

INSERT INTO commerce_data.Product (prod_id, title) VALUES ('3', 'Ceai');
```

```
INSERT INTO commerce_data.Product (prod_id, title) VALUES ('4', 'Cafea');
```

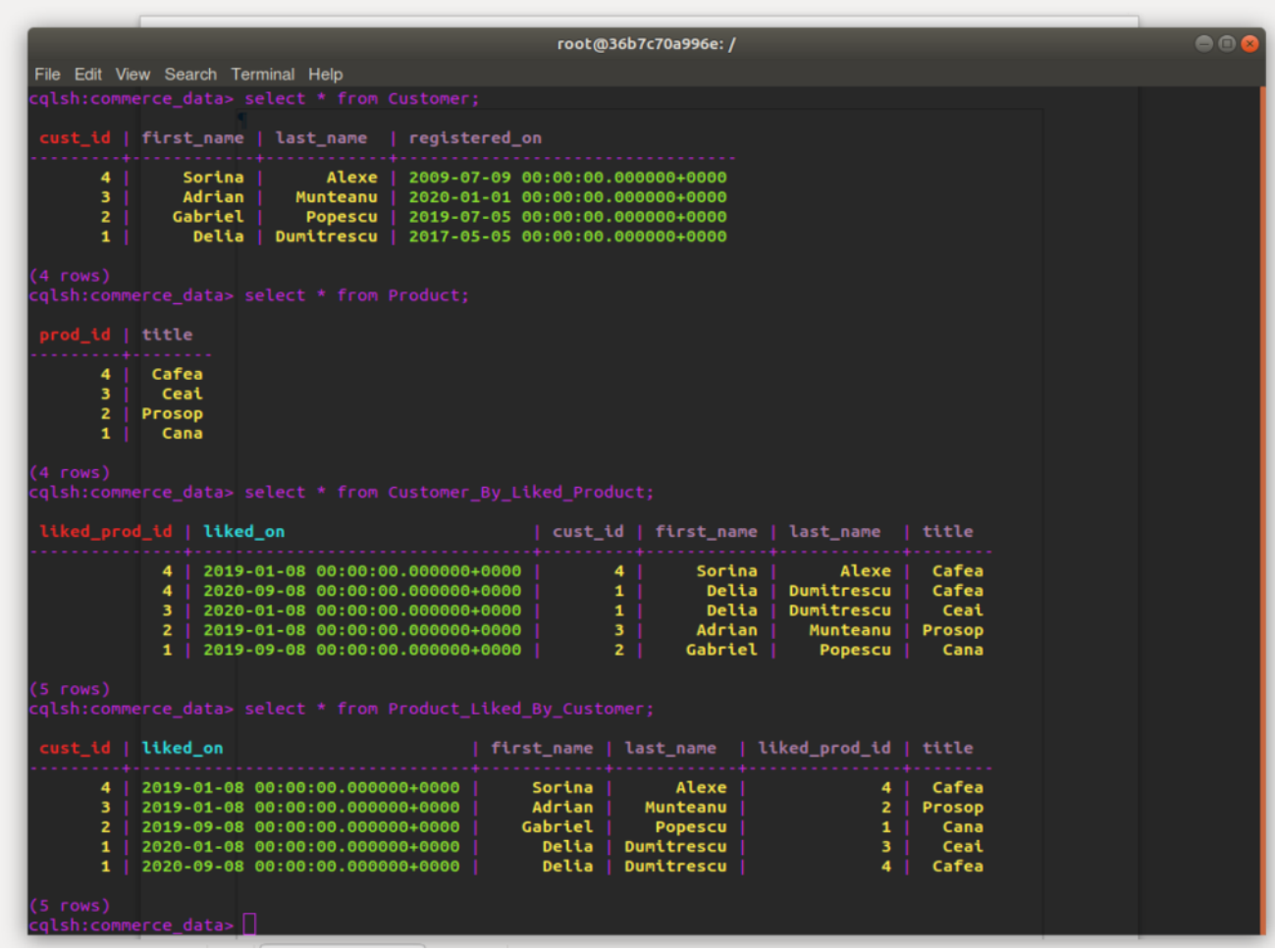
Tabelul Customer_By_Liked_Product:

```
INSERT INTO commerce_data.Customer_By_Liked_Product (liked_prod_id, liked_on,
title, cust_id, first_name, last_name) VALUES ('4', '2019-01-08', 'Cafea',
'4', 'Sorina', 'Alexe');

INSERT INTO commerce_data.Customer_By_Liked_Product (liked_prod_id, liked_on,
title, cust_id, first_name, last_name) VALUES ('2', '2019-01-08', 'Prosop',
'3', 'Adrian', 'Munteanu');

INSERT INTO commerce_data.Customer_By_Liked_Product (liked_prod_id, liked_on,
title, cust_id, first_name, last_name) VALUES ('1', '2019-09-08', 'Cana',
'2', 'Gabriel', 'Popescu');

INSERT INTO commerce_data.Customer_By_Liked_Product (liked_prod_id, liked_on,
title, cust_id, first_name, last_name) VALUES ('3', '2020-01-08', 'Ceai',
'1', 'Delia', 'Dumitrescu');

INSERT INTO commerce_data.Customer_By_Liked_Product (liked_prod_id, liked_on,
title, cust_id, first_name, last_name) VALUES ('4', '2020-09-08', 'Cafea',
'1', 'Delia', 'Dumitrescu');
```

Tabelul Product_Liked_By_Customer:

```
INSERT INTO commerce_data.Product_Liked_By_Customer (cust_id, first_name,
last_name, liked_prod_id, liked_on, title) VALUES ('1', 'Delia', 'Dumitrescu',
'3', '2020-01-08', 'Ceai');

INSERT INTO commerce_data.Product_Liked_By_Customer (cust_id, first_name,
last_name, liked_prod_id, liked_on, title) VALUES ('1', 'Delia', 'Dumitrescu',
'4', '2020-09-08', 'Cafea');

INSERT INTO commerce_data.Product_Liked_By_Customer (cust_id, first_name,
last_name, liked_prod_id, liked_on, title) VALUES ('2', 'Gabriel', 'Popescu',
'1', '2019-09-08', 'Cana');

INSERT INTO commerce_data.Product_Liked_By_Customer (cust_id, first_name,
last_name, liked_prod_id, liked_on, title) VALUES ('3', 'Adrian', 'Munteanu',
'2', '2019-01-08', 'Prosop');

INSERT INTO commerce_data.Product_Liked_By_Customer (cust_id, first_name,
last_name, liked_prod_id, liked_on, title) VALUES ('4', 'Sorina', 'Alexe',
'4', '2019-01-08', 'Cafea');
```

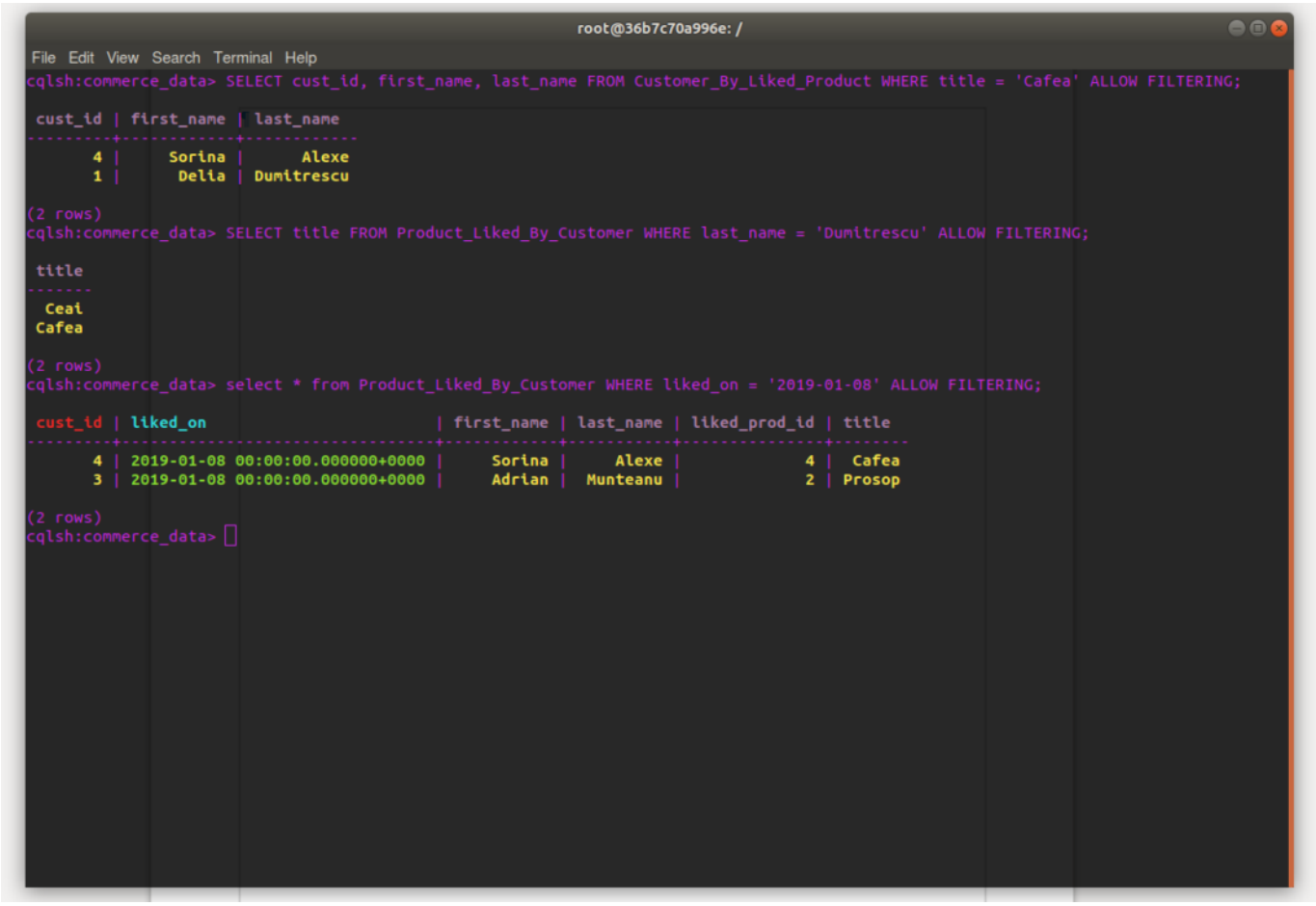
Dupa executarea tututor comenzilor, tabelele generate for fi:



4.5 Executarea unor query-uri

- Numele prenumele si id-ul clientilor carora le-a placut produsul 'Cafea'
- Numele produselor pe care clientii cu numele de familie "Dumitrescu" le-au apreciat
- Toate datele despre produsele apreciate in data de '2019-01-08'

Rezultatele query-urilor sunt:



5 Final

Bazele de date de tip *NoSQL* reprezinta un alt mod de a stoca date. Fie ca vine vorba de *Apache Cassandra*, *MongoDB* sau *Redis*, toate au la urma urmei acelasi scop: gasirea unei solutii pentru a stoca cantitati foarte mari de date dinamice ce pot fi accesate foarte usor. Alegerea unei baze de date trebuie facuta pe baza nevoilor proiectului la care sunt folosite. Exista cazuri in care bazele de date *NoSQL* sunt superioare celor *SQL*, dar si cazuri in care alegerea unei astfel de baze de date ar fi un dezastru. La urma urmei, ele satisfac nevoi diferite.