Extragerea cunoștințelor din baze de date – Proiect

Extragerea de cunoștințe în contextul unei aplicații de comunicare centralizată

Student: **Ionuț-Alexandru BALTARIU**
Cadru didactic coordonator: **Asist. dr. ing. Cătălin MIRONEANU**

Descrierea proiectului

Colecțiile din cadrul proiectului sunt o formă simplificată de reprezentare a datelor care sunt tipice unei aplicații de comunicare online (e.g: *Slack*). Aceasta aplicație are rolul de a facilita comunicarea centralizată pe mai multe canale canale posibile a tuturor utilizatorilor unei organizații. În cadrul aplicației, fiecare utilizator are mai multe date personale care sunt expuse tuturor celorlalți (e.g: *nume, prenume, email, funcție* etc). De asemenea, fiecare utilizator poate lăsa comentarii în canalele pe care acesta este înscris, acestea putând să obțină reacții din partea celorlalți (eng. *likes*).

Pentru popularea colecțiilor cu un număr considerabil de date s-a folosit limbajul `Python`, mai exact modulul `Faker`, care permite generarea mai multor date de interes(e.g: *nume, prenume, job, mâncare preferată, fragmente de text etc*). Existența unui număr ridicat de date a dus la înțelegerea mai în detaliu a operațiunilor posibile în bazele de date nonrelaționale MongoDB, întrucât rezultatele vizualizate în urma operațiunilor de filtrare/agregare reflectă realitatea.

Structura colecțiilor

În cadrul proiectului există 3 colecții:

1. **channels** - canalele existente în contextul aplicației
2. **comments** - comentariile lăsate de utilizatori, acestea sunt corelate cu un canal și un utilizator specific
3. **users** - utilizatorii finali ai aplicației, aceștia pot fi "abonați" la variate canale și pot lăsa comentarii (tot în cadrul canalelor la care sunt abonați). De asemenea, utilizatorii expun o serie de date personale tuturor celorlalți membri din cadrul aplicației

Structura efectivă a colecțiilor este următoarea:

Channel

```
{
    "bsonType": "object",
    "required": ["name", "type", "meta", "creation_date"],
    "properties": {
        "name": { "bsonType": "string" },
        "type": { "bsonType": "string" },
        "creation_date": { "bsonType": "date" },
        "meta": {
            "bsonType": "object",
            "required": ["archived"],
            "properties": {
                "archived": { "bsonType": "bool" }
            }
        }
```

```
    }
}
```

Comment

```
{
    "bsonType": "object",
    "required": ["value", "channel", "user", "likes", "sent_at"],
    "properties": {
        "value": { "bsonType": "string" },
        "channel": { "bsonType": "objectId" },
        "user": { "bsonType": "objectId" },
        "sent_at": { "bsonType": "date" },
        "likes": { "bsonType": "int" }
    }
}
```

Users

```
{
    "bsonType": "object",
    "required": ["first_name", "last_name", "user_name", "email", "ex_jobs",
                 "current_job", "registration_date",
                 "personal_information", "channels"],
    "properties": {
        "first_name": { "bsonType": "string" },
        "last_name": { "bsonType": "string" },
        "user_name": { "bsonType": "string" },
        "email": { "bsonType": "string" },
        "registration_date": { "bsonType": "date"},
        "ex_jobs": {
            "bsonType": "array",
            "items": {
                "bsonType": "object",
                "required": ["name", "company"],
                "properties": {
                    "name": { "bsonType": "string" },
                    "company": { "bsonType": "string" }
                }
            },
        },
        "current_job": {
            "bsonType": "object",
            "required": ["name", "years_of_experience"],
            "properties": {
                "relationship_status": { "bsonType": "string" },
                "years_of_experience": { "bsonType": "int" }
            }
        },
        "personal_information": {
            "bsonType": "object",
            "required": ["relationship_status", "zodiac", "favorite_food"],
```

```
            "properties": {
                "relationship_status": { "bsonType": "string" },
                "zodiac": { "bsonType": "string" },
                "favorite_food": { "bsonType": "string" }
            }
        },
        "channels": {
            "bsonType": "array",
            "items": { "bsonType": "objectId"},
        }
    }
```

Se observă faptul că `Users` și `Comments` conțin, printre câmpurile obișnuite și referințe către celelalte colecții. Acest detaliu permite atât relaționarea efectivă a documentelor JSON aferente colecțiilor, precum și afișarea imbricată în cadrul agregărilor (*look-up*).

Descrierea funcționalităților și a operațiilor corespunzătoare din fișierele de script

Proiectul conține următoarele scripturi `Python` și `Javascript`, cu mențiunea că inclusiv operațiunile MongoDB au fost executate din interiul acestora, folosind conectorul `PyMongo`. Acesta oferă toate funcționalitățile necesare pentru îndeplinirea cerințelor proiectului, însă, din cauza complexității scrierii operațiilor map-reduce în PyMongo, s-a ales scrierea acestora în fișiere separate.

```
.
├── bulk_ops.py
├── create.py
├── delete.py
├── docker-compose.yml
├── fake_data_gen.py
├── map_reduce_1.js
├── map_reduce_2.js
├── map_reduce_3.js
├── mongo_handler.py
├── more_complex_searches.py
├── simple_searches.py
└── update.py
```

Figura 1: Scripturile din cadrul proiectului

mongo_handler.py

Este un utilitar în care se definește un `wrapper` peste obiectele corespunzătoare bazelor de date și colecțiilor din cadrul modulului `PyMongo`. Utilitarul facilitează reîncercarea conectării la server-ul MongoDB în cazul în care aceasta eșuează din varii motive.

fake_data_gen.py

Utilitar care are simplul rol de a facilita generarea datelor realiste într-un mod aleator.

create.py

Acest script are rolul de a insera datele inițiale: 19 canale, 500 de utilizatori cu date generate aleator (însă realiste) și 15000 de comentarii (de asemenea, generate aleator).De asemenea, în acest script este creat și un index de tip `Text` pe câmpul `first_name` din colecția `users`. Implicit, la

inserarea datelor, se crează și colecțiile descrise în secțiunea anterioară. După cum s-a menționat, pentru generarea datelor s-au folosit modulele `Faker` și `random`.

Au fost folosite instrucțiunile următoare: `insertMany, find` (pentru abonarea utilizatorilor la un număr aleator de canale - obținerea id-urilor acestor) și `createIndex`.

update.py

După cum îi sugerează și numele, scriptul conține o serie de instrucțiuni care se ocupă cu actualizarea documentelor deja stocate în MongoDB. Actualizările au fost făcute cu variate filtre, pentru ca acestea să se reflecte doar în cazurile dorite. Comenzile MongoDB folosite în acest fișier sunt: `updateOne` și `updateMany`.

delete.py

Similar operațiunilor de actualizare, operațiunile de ștergere se execută doar pentru acele documente care respectă condițiile impuse in filtre. Mai mult, este posibilă ștergerea tuturor documentelor din toate colecțiile, operațiune făcută prin omitirea intenționată a unor filtre de ștergere. Pentru a realiza ștergerea tuturor documentelor, trebuie să se paseze un argument in linie de comandă ce conține următoarele caractere: `"all"`. Singura operație MongoDB folosită în acest fișier este `deleteMany`, întrucât alternativa orientată pe un singur document (`deleteOne`) nu ar fi produs un efect sesizabil.
Exemplu execuție script: `python3 delete.py all`

bulk_ops.py

Scriptul conține variate operații `bulk`, mai exact, operații complexe compuse din mai multe suboperații care pot diferi una față de cealaltă și care sunt executate în ordinea scrierii. Printre altele, o instrucțiune de tip `bulk` poate conține: *insertOne, updateOne, deleteOne, updateMany, deleteMany* etc. Aceste instrucțiuni sunt executate secvențial, astfel, în cazul în care o eroare este întâmpinată undeva în secvența de operații, se oprește întreaga execuție, fără a se face `rollback`.

Echivalentul MongoDB al operației descrise este `bulkWrite`, cu mențiunea că a fost folosită varianta în care suboperațiile sunt executate ordonat (`ordered=True`, parametru implicit).

simple_searches.py

Sunt regăsite variate operații de căutare a documentelor din colecțiile descrise anterior. Au fost folosite expresii regulate, indecși de tip text, filtrări pe câmpuri imbricate, operatori specifici MongoDB (`$gte,$or`), sortări, proiecții, limite și omiteri de date (`limit, skip`) etc. Singura instrucțiune MongoDB folosită este `find`.

more_complex_searches.py

Corespunzător penultimei cerințe din cadrul proiectului, scriptul conține variate operațiuni de căutare și agregare complexă. Printre operatorii MongoDB folosiți se regăsește și `lookup`, care are rolul de a înlocui dinamic identificatorii altor colecții referite (relații) cu reprezentarea acestora. Se pot forma, astfel, documente mai complexe care ilustrează efectiv relațiile colecției respective. Un exemplu de agregare pentru un document de tip **comment** este următorul:

```
{
    "_id" : ObjectId("6274201412938483fda70908"),
    "value" : "Aliquam vitae laborum ullam rerum voluptas.",
    "channel" : [
        {
```

```
            "_id" : ObjectId("6274201412938483fda7070c"),
            "name" : "#shoutouts",
            "type" : "public",
            "creation_date" : ISODate("2020-01-06T10:54:39.000+0000"),
            "meta" : {
                "archived" : false
            }
        }
    ],
    "user" : [
        {
            "first_name" : "Camil",
            "last_name" : "Gheorghiu"
        }
    ]
}
```

Se observă că în documentul corespunzător colecției **comments** se regăsesc subdocumentele care detaliază canalul, respectiv utilizatorul corelat cu acel comentariu. De asemenea, au fost folosiți operatori precum *$unwind, $group, $project, $sort*.

map_reduce_1.js & map_reduce_2.js & map_reduce_3.js

- **map_reduce_1.js** - calcularea numărului de aprecieri în contextul unui canal, după data de 01.01.2022
- **map_reduce_1.js** - calcularea numărului de canale la care sunt abonați utilizatorii care au cel puțin doi ani de experință la jobul curent
- **map_reduce_1.js** - calcularea unui scor (cu valoarea între 1 și 10) corespunzător numărului de utilizatori abonați unui canal. Pentru 500 utilizatori (plafon) abonați se obține numărul 10, iar pentru 0 utilizatori abonați se obține numărul 0.

De menționat este faptul că scripturile map-reduce au fost realizate cu ajutorul utilitarului din cadrul `Studio 3T`, care permite editarea individuală a funcțiilor de `map, reduce, finalize` și adăugarea unor condiții variate de filtrare, sortare etc.