Considerații asupra bazelor de date NoSQL

Dragoş NICOLAU

Institutul Naţional de Cercetare-Dezvoltare în Informatică, ICI Bucureşti

dragos.nicolau@ici.ro

Rezumat: Lucrarea de faţă şi-a propus să prezinte câteva consideraţii asupra bazelor de date ne-relaţionale, (denumite NoSQL) utilizate din ce în ce mai mult mai ales de către aplicaţiile (în special Web, cum ar fi Youtube, Amazon, Google şi Facebook) care trebuie atât să stocheze un volum impresionant de date, cât şi să ofere, pe baza acestor date, răspunsuri rapide la interogări concurente, cu nivel de ritmicitate tot mai ridicat. Sunt prezentate principalele avantaje ale bazelor NoSQL, împreună cu o mostră de sintaxă corectă pentru gruparea unui ansamblu de date pregătit pentru a fi inserat cu succes într-o bază NoSQL şi sunt ilustrate câteva modele de realizare a conexiunilor; este trecut în revistă sistemul MongoDB şi sunt oferite exemple concrete de interogări, inclusiv comparative (SQL vs NoSQL). Contribuţia autorului constă în (a) prezentarea unui nou concept informatic şi (b) utilizarea acestuia într-un concret proiect de cercetare aplicată.

Cuvinte cheie: Baze de date NoSQL, Baze de date relaţionale.

Considerations on NoSQL Databases

Abstract: This paper aims to present some considerations on non-relational databases (called NoSQL) that are increasingly being used by applications (especially the Web, such as Youtube, Amazon, Google and Facebook) that both have to store an impressive volume of data, and to provide, on the basis of these data, quick responses to concurrent queries with rising rhythm levels. A correct syntax sample is presented for grouping a set of data ready to be successfully inserted into a NoSQL data base, and illustrated with several examples of how to make connections through concrete examples; comparative samples of queries are also provided(SQL vs NoSQL).

Keywords: NoSQL Databases, Relational Databases.

1. Introducere

Organizaţiile (companii, instituţii, agenţii etc.) colecţionează, în format electronic, cantităţi apreciabile de date, utilizabile în diverse scopuri: comercializare de produse, oferire de informaţii de presă sub format text sau multimedia, furnizare de programe software [6], prezentare de conţinuturi cu caracter financiar, juridic sau administrativ, deservirea de platforme software pentru telemedicină [4] sau realizare de prognoze şi analize pentru propriile strategii de lucru etc. În mod tradiţional, aceste date se stochează în baze de date relaţionale. Precizăm că baza relaţională (clasică) este o colecţie fixă de tabele; tabelul este o colecţie fixă de câmpuri, în dreptul fiecărui câmp memorându-se o listă - teoretic, oricât de lungă - de date de exact acelaşi tip; între tabele se pot stabili conexiuni, fie sub forma unor câmpuri suplimentare - care într-un tabel de date memorează identificatori unici din alt tabel de date - fie sub forma unor tabele de legătură; câmpul reprezintă o categorie informaţională, adică un segment (identificabil prin nume) de informaţie de un anume tip, bine determinat. În ultima vreme, însă, mulţi dezvoltatori au început să implementeze şi să propună spre utilizare baze de date non-relaţionale, numite NoSQL ("Not Only SQL – Nu Doar SQL"). Odată cu dezvoltarea şi răspândirea aplicaţiilor Web (se disting printr-un pronunţat caracter interactiv), a apărut necesitatea stocării unei cantităţi de informaţie al cărei volum creşte tot mai rapid: utilizatorii nu se mai mulţumesc doar cu rolul de a interoga bazele de date, ci doresc inclusiv să producă conţinut (prin trimitere de texte sau fişiere de orice tip).

Universul Web a impus regândirea modalităţii de stocare şi procesare a datelor, astfel că s-a purces la căutarea de soluţii concrete pentru construirea unor mecanisme software care să ofere viteză sporită la analizarea unui număr tot mai mare de înregistrări. Bazele de date deservind aplicaţii Web nu se confruntă cu interogări complexe (în general, izvorâte dintr-o multitudine de asocieri *inner* – echivalente cu intersecţia unor colecţii de date şi *outer* – echivalente cu reuniunea unor colecţii de date), ci mai degrabă cu interogări simple, dar solicitante în ceea ce priveşte volumul şi ritmicitatea. Bazele de date NoSQL răspund cu succes cerinţelor de volum şi viteză mai sus amintite.

Confruntate cu sporul de trafic şi de volum stocat, bazele de date relaţionale nu mai prezintă eficienţă din perspectiva vitezei, în special în cazul aplicaţiilor care gestionează volume colosale de date. Printre primele aplicaţii (cu rulaj impresionant de date) care au ridicat problema adaptabilităţii la efort crescut se număra Facebook, Google şi Youtube. Conform studiilor estimative, volumul de date stocate în format electronic înregistrează o creştere rapidă de la an la an, în 2018 atingând nivelul de aproape 15 000 exabytes. [2] (1 EB = 10^{18} octeţi = 1 miliard de miliarde de octeţi).

Scopul articolului este acela de a prezenta aspecte generale / avantaje ale bazelor NoSQL şi modalitatea de structurare - conectare a datelor, de a trece în revistă sistemul MongoDB împreună cu o platformă de versionare de fişiere la care a fost utilizat, de a ilustra modalităţile concrete de efectuare a interogărilor. Contribuţia autorului constă în (a) - prezentarea unui nou concept informatic şi (b) - utilizarea acestuia într-un concret proiect de cercetare aplicată.

2. Aspecte generale referitoare la bazele de date NoSQL

Principalul avantaj al bazelor NoSQL este acela că permit interogări eficace datorită faptului că datele (oricât de complex ar fi organizate) sunt memorate în format cu structurare "elastică" (fără schemă şi fără tabele de conexiune; precizăm că prin schemă se înţelege - **stricto sensu** - organizarea informaţiei în tabele prin departajare în câmpuri cu nume propriu şi tip fix de date). În prezent, există baze de date NoSQL dezvoltate de multe companii (cum ar fi Amazon şi Google) ale căror aplicaţii Web procesează cantităţi uriaşe de date.

Conform unui studiu (*"A Comparison Between MongoDB and MySQL Document Store Considering Performance"* - din anul 2017) dedicat comparării vitezei de lucru între baze NoSQL şi cele MySQL, cele 2 sisteme de gestiune dezvoltă viteze comparabile la operaţiunile de actualizare şi interogare într-un tabel, dacă numărul de înregistrări este de până la 100. 000; când "populaţia" de înregistrări depăşeşte acest număr, sistemele NoSQL sunt capabile să dezvolte viteză de procesare sensibil mai mare, aceasta fiind până la de 8-10 ori superioară celei dezvoltate de MySQL, când numărul de înregistrări frizează milionul. Astfel, se poate observa faptul că NoSQL devine un concurent serios pentru MySQL (sistem relaţional, a cărui principală virtute unanim recunoscută este viteza) [5].

Rezumând, principalele trăsături funcţionale ale bazelor NoSQL sunt:

- Lipsa schemei.
- Simplitate anatomică, ceea ce permite executarea de interogări rapide.
- Posibilitatea de a adăuga / edita / şterge dinamic noi atribute (câmpuri) la înregistrările existente.
- Posibilitatea de a partaja încărcarea pe mai multe servere (adică distribuirea "efortului" de calcul).
- Posibilitatea de a replica datele pe mai multe servere.
- Accesare simultană posibil problematică faţă de modelul relaţional (negarantarea tranzacţiilor).

Din perspectiva dezvoltatorilor, bazele NoSQL sunt open-source şi se bucură de colecţii generoase de funcţii API pentru dialogul cu aplicaţii scrise în diverse limbaje. Sistemele NoSQL se bucură de foarte bună compatibilitate cu tehnologia Cloud, care se bazează pe virtualizare.

Principalele argumente pentru dezvoltarea şi utilizarea de baze de date NoSQL sunt:

- Necesitatea satisfacerii unui trafic masiv de date.
- Evitarea complexităţii morfologice prin stocarea datelor după modelul structurilor din limbajele de programare, nu după modelul relaţional. Cele mai multe aplicaţii operează cu structuri de date masive dar de complexitate relativ scăzută.
- Duplicare şi partajare uşor de executat.
- Sacrificarea minoră a fiabilităţii în favoarea creşterii performantei, aşa cum e cazul siturilor de socializare.
- Comoditate la programarea aplicaţiilor care interacţionează cu baze de date.

3. Exemplu de organizare a datelor în Baze de Date NoSQL

3.1. O nouă anatomie, noi performanţe

După cum am precizat, bazele NoSQL reprezintă o nouă modalitate de stocare a informaţiei, fără schemă şi fără tabele de conexiune, înregistrările fiind structuri arborescente de tip JavaScript.

Această nouă morfologie oferă trei avantaje esenţiale: viteză crescută la interogarea seturilor de date, uşurinţă în programare şi capabilitatea stocării unei cantităţi uriaşe de informaţie [3]. Reversul îl constituie lipsa garanţiei ferme că o tranzacţie a fost procesată corect (riscul minor de a accesa date care fie nu mai există, fie au suferit modificări neprevăzute), aceste disfuncţii potenţiale fiind absolut excluse în cazul bazelor de date clasice (relaţionale), unde operaţiile sunt de tipul "tot sau nimic".

Interogările în NoSQL se bazează în general pe căutari bazate pe chei primare (sau pe un câmp ID), pe lipsa tabelelor de legătură şi pe "investigarea" unui număr mic de tabele - ceea ce duce la un impresionant spor de viteză (de exemplu, în modelul NoSQL, scrierea într-o bază de date de 50 GB este mai rapidă de 2.500 ori faţă de tradiţionalul MySQL, el însuşi un sistem recunoscut pentru viteza de lucru).

Capabilitatea de a stoca volume imense de date, o altă caracteristică importantă a bazelor de date NoSQL, rezidă în distribuirea "pe orizontală" a efortului de calcul. Distribuirea "pe orizontală" înseamnă rularea mai multor instanţe **identice** pe servere diferite, simultan. Este utilă în situaţiile reclamând trafic foarte mare şi satisfacerea unui număr apreciabil de cereri simultane. Soluţiile NoSQL adaptabile la creşterea efortului [10] (calitate denumită în mod anti-intuitiv şi neglijent în literatura anglo-saxonă cu nepotrivitul termen de *"scalability"*, care sugerează puternic posibilitatea de măsurare, nicidecum pe cea de uniformizare a încărcării). Ca un plus de contribuţie ştiinţifică, propunem introducerea în lexicul informatic românesc a termenului de "para-încărcabilitate".

Printre sistemele NoSQL din ziua de azi pot fi menţionate Hbase, MongoDB, CouchDB, GTM [9]. Memorarea datelor poate fi uşor organizată în modele oricât de complexe, ceea ce subliniază flexibilitatea crescută oferită de modelul NoSQL.

3.2. Mostre sintactice

Mai jos se prezintă o mostră de sintaxă a unui **document apt de a fi inserat** (omologul **înregistrării** - record). Se permit câmpuri omonime **doar dacă** sunt imbricate. Sintaxa, care este de tip JSON (*JavaScript Object Notation* - reguli de alcătuire pentru structuri de date şi clase în JavaScript, notaţie aleasă tocmai pentru că reprezintă formula optimă de reprezentare arborescentă

a informaţiei), **nu** permite "fraţi" omonimi [7]. Fiecare document este încadrat de acolade, separarea dintre câmp şi valoarea asociată se face prin ":", valoarea fiecărui câmp de tip structură este delimitată de acolade, valorile fiecărui câmp de tip vector (listă) sunt încadrate de paranteze pătrate, perechile câmp: valoare sunt separate prin virgulă. Desigur, exemplul de mai jos este aberant într-o logică strict utilitaristă, însă edificator şi destul de cuprinzător în ceea ce priveşte atât rigoarea cât şi generozitatea sintactică.

```
{
     "acest cimp are valoare de tip text": "un text",
     "acest cimp are valoare de tip intreg": 25,
     "enumerare de texte": ["un text", "alt text"],
     "enumerare de valori de orice tip (accesabile prin propriul index)":
     [
          "text oarecare", {} ,
          "alt text oarecare",
         {
              "cimp": 3.14, "alt cimp": "valoare",
              "stat": {"Nume":"RO", "judete": ["AG", "CT", "IF", "VL"]}
         }, 3.14 , [], {"Statiune" : "NEPTUN"}, [],
     ],
     "nume obligatoriu aici, chiar daca lista este initial vida": [],
     "nume obligatoriu aici, chiar daca struct este initial vida": {}
}
```

Numele de câmpuri sunt texte (string) care trebuie să respecte anumite reguli. Identificatorul **_id** este generat şi inserat automat (de către sistemul de gestiune) dacă nu este prezent în structura documentului bun de inserat (**"_id": ObjectId("5146bb52d8524270060001f3")**), altminteri este păstrat ca atare (o opţiune puternic descurajată), evident dacă nu reprezintă un duplicat. în exemplul de mai sus se poate observa elasticitatea (versatilitatea) modelului de organizare a datelor, el putând suporta practic orice combinaţie de vectori, structuri de date şi date native (numere reale, numere întregi, texte).

Este important de menţionat că bazele NoSQL suportă şi tipul de date BLOB (*Binary Large Object* - obiect binar de mari dimensiuni), ceea ce face posibilă memorarea tale-quale a fişierelor, ca sub-entităţi (denominalizabile după voie) ale **secţiunii separate GridFS** a bazei, dedicate special doar lor, fiecare fişier putând fi asociat ulterior cu orice fel de combinaţie de date de interes (metadate).

Omologul **tabelului (***table***)** este **colecţia**, ea cuprinzând lista de documente.

3.3. Mostre de conexiuni

Baza de date NoSQL nu stochează conexiuni folosind tabele specializate, ci foloseşte chei de identificare chiar în cadrul înregistrărilor (documentelor), aşa cum vom vedea imediat mai jos. Acest tip de organizare conservă performanţele la schimbările ce pot surveni de-a lungul timpului în alcătuirea bazei. într-un sistem relaţional nu există flexibilitatea necesară pentru a executa modificări în modelul de date. Spre diferenţă, lipsa schemei fixe (rigide) conferă bazelor NoSQL o remarcabilă versatilitate, dând astfel posibilitate dezvoltatorilor ca la nevoie să opereze schimbări de structură "din mers" în deplin comfort (fără schimbări majore în cod). Desigur, preţul plătit pentru sporul de viteză rezultat din eliminarea **tabelelor de conexiune (legătură)** îl reprezintă redundanţa prezenţei câmpurilor de tip ID în documente (trebuie spus că utilizarea tabelelor de corespondenţă nu este imposibilă/interzisă funcţional, ci puternic deconsiliabilă; la rigoare, nimeni nu opreşte pe dezvoltator să le utilizeze). Conexiunile între colecţii pot încarna toate combinaţiile posibile.

Primul exemplu este varianta arborescenţei: fiecare document din Tabelul TARA conţine un câmp de tip vector cu toate ID judeţ, fiecare document din Tabelul JUDET conţine un câmp de tip

vector cu toate ID localitate, fiecare document din Tabelul LOCALITATE conţine un câmp de tip vector toate cu ID muzeu. Pentru accelerarea interogărilor de tip "în ce ţară se află Muzeul de Istorie X ?", fiecare document poate memora ID propriului "părinte" (de ex., fiece document localitate va memora câte un singur ID judeţ). Certamente, exemplul este aberant din punct de vedere utilitaristic (în plus, în mod ironic, însuşi spiritul NoSQL ar impune în colecţia MUZEU identificatori de oraş, judeţ şi ţară), dar edificator pentru ilustrarea noilor posibilităţi de conectare între colecţii (tabele).

Al doilea exemplu este o conexiune *many-to-many*, dintr-o ipotetică bază "LICEU", redat în figura de mai jos: la o clasă predau mai mulţi profesori, un profesor predă la mai multe clase.

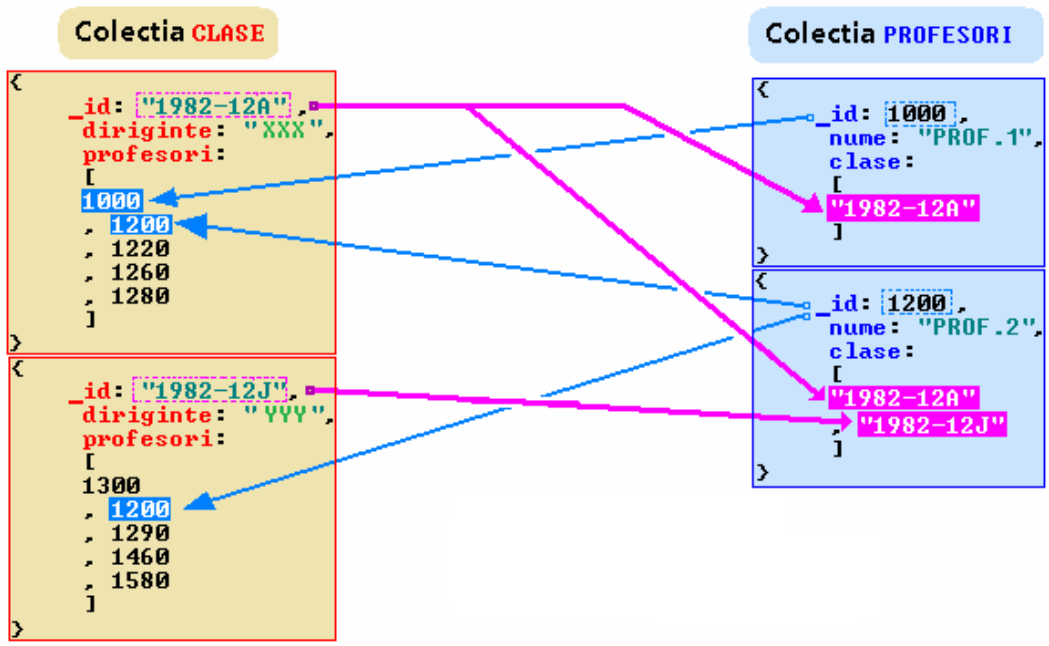

Figura 1. Exemplu de conexiune many-to-many

Valorile de tip ID au fost alese astfel încât să fie ilustrative pentru ideea de conectare a colecţiilor, valoarea corectă fiind un unic string format din 24 de cifre hexazecimale (adică **0 : F**) generat aleator.

4. Sistemul MongoDB. Exemplu de utilizare la un proiect concret

4.1. Sistemul MongoDB

MongoDB este un sistem de gestiune de baze de date open-source NoSQL scris în C++, care lucrează cu date nestructurate (în sens clasic), organizându-le în format bloc-arborescent. MongoDB asigură performanţe avansate, disponibilitate ridicată şi foarte bună adaptare la efort de calcul crescut [1].

MongoDB este prevăzut în mod implicit cu o aplicaţie client de tip consolă şi anume executabilul **bin/mongo.exe**, reprezentând *shell*-ul (executabil client, tip Consolă, care analizează şi trimite comenzile spre interpretare Serviciului Windows Mongo – încarnat de **bin/mongod.exe** - şi de la care va recolta apoi rezultatele returnate) interactiv scris în C/C++ şi apt de a interpreta instrucţiuni JavaScript. *Shell*-ul este folositor pentru verificări teste și pentru funcţiile administrative (textual, *"Shell"* înseamnă "carapacea care încapsulează comenzi"). Fiecare bază are ataşate două fişiere (**BazaOarecare.0** şi **BazaOarecare.ns**) amplasate într-un folder dedicat, configurabil după instalare, prin editarea unui fişier de iniţializare.

4.2. Sistemul de versionare

În cele ce urmează vom prezenta în mod concret utilizarea sistemului MongoDB la alcătuirea unei baze de date care deserveşte o aplicaţie de versionare a codurilor sursă folosite la dezvoltarea de proiecte software. Baza poartă numele companiei de software care va utiliza acest depozit de cod. În componenţa bazei există următoarele colecţii (tabele):

- **Proiecte;** fiecare proiect conţine date proprii şi o listă cu numele dezvoltatorilor care participă la elaborarea lui, un dezvoltator putând participa la mai multe proiecte simultan. Numele fiecărui proiect este unic în cadrul bazei (Companiei).
- **Dezvoltatori;** fiecare dezvoltator are ataşate date proprii (parolă, drepturi etc.) şi are nume unic în cadrul companiei, de aceea acest tabel (colecţie) a fost înzestrat cu restricţia de a nu accepta duplicate pentru câmpul nume. Astfel, orice tentativă de a insera un dezvoltator omonim cu unul deja existent va emite un string de eroare.
- **Fişiere;** fişierele fiind informaţii de tip BLOB (*Binary Large Object* – obiect binar de mari dimensiuni), vor fi stocate automat într-o sub-secţiune (care poate fi denumită după voie) a zonei consacrate de tip *GridFileSystem*. Fiecare intrare de tip BLOB are nume propriu şi are ataşat un segment propriu de tip metadate, unde se memorează sub formă de JSON toate elementele de interes:
    - proiectul (_id) căruia aparţine fişierul curent.
    - dezvoltatorul (_id) care a încărcat fişierul curent.
    - versiunea MAJORĂ: string de la "0" la "9".
    - versiunea minoră: string de la "0" la "9".
    - comentariu opţional, de tip text.
    - şirul indecşilor liniilor care nu mai concordă exact cu cele respectiv de acelaşi index din versiunea imediat precedentă.
    - indicativ că versiunea este finală (versiunea următoare trece în următoarea treaptă MAJORĂ.

  Numele fişierelor nu trebuie să fie unice.

Figura de mai jos ilustrează conexiunea proiect -> dezvoltator prin inserarea tuturor numelor dezvoltatorilor participanţi în câmpul de tip vector {**`Proiect.Dezvoltatori`**} al colecţiei Proiect, precum şi conexiunile fisier->proiect şi fisier->dezvoltator prin utilizarea secţiunii de metadate a colecţiei de tip BLOB (tabelul care memorează fişiere).

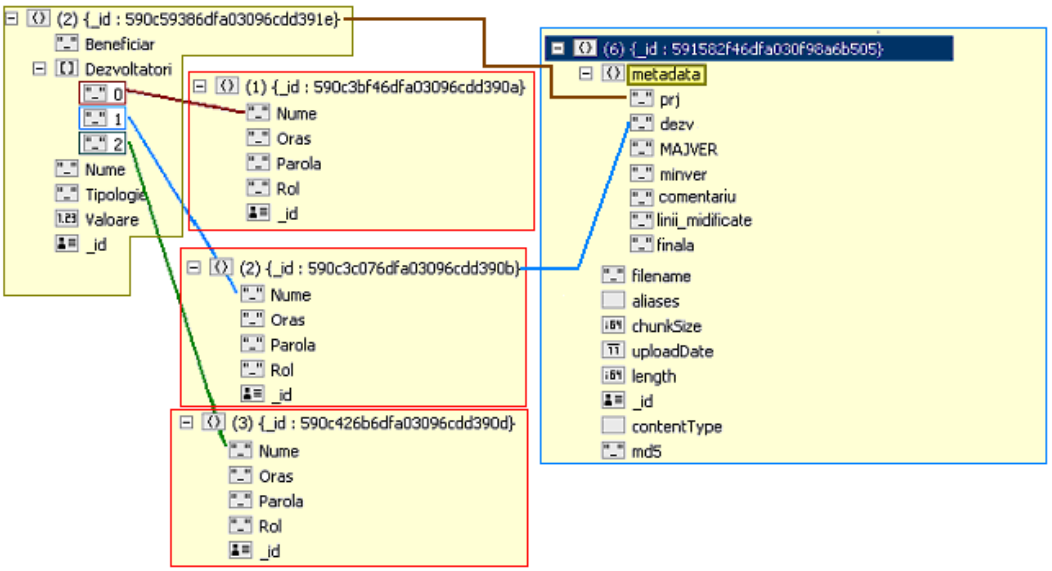

Figura 2. Conexiuni între colecţii (Proiect-Dezvoltator, Proiect-Fişier, Dezvoltator-Fişier)

Trebuie menţionat că mărimea (volumul) unei baze de date utilizate într-un sistem de versionare creşte foarte repede, fiindcă e de presupus că un număr important de dezvoltatori încarcă des fişiere – aceasta în condiţiile în care, în mod normal, nimic nu se şterge!

Din acest motiv, soluţia NoSQL este indiscutabil preferabilă, mai ales că sistemul de gestiune MongoDB are capabilitatea de a diviza comod o bază de date în sub-componente (*"shards"*) care se pot eventual transfera pe alte maşini, unde se vor afla sub gestiunea altor instanţe fizice a serverului de baze de date.

În această situaţie, conjugarea soluţiei NoSQL cu tehnologia Cloud reprezintă varianta optimă de gestiune şi lucru, ţinând cont de faptul că maşinile auxiliare sus-amintite rezidă pe exact aceeaşi platformă fizică de rulare, adică aceea pusă la dispoziţie de unitatea Cloud!

În tabelul de mai jos se exemplifică prezenţa în subsecţiunea denumită "SURSE" a zonei **GridFS** a două versiuni consecutive ale aceluiaşi fişier, din acelaşi proiect, dezvoltat de către aceeaşi persoană, fiecare variantă având propriul conţinut fizic, binar. [8]

Toate componentele de tip binar (BLOB) ale unui document (înregistrare) sunt memorate, în universul NoSQL, nu în vreun tabel, ci într-o **zonă** dedicată a bazei. Precizăm că `_idPrj1` şi `_idDev` joacă mai jos rol pur ilustrativ, în realitate fiind de forma `ObjectId("5146bb52d8524270060001f3")`.

Tabel 1. Ilustrarea memorării fişierelor într-o subsecţiune a zonei GridFileSystem a bazei de date

Name	id	Metadata
fisier14.cpp	5915821e6dfa030f98a6b4fb	{ "prj" : _idPrj1, "dezv" : _idDev, "MAJVER" : "0", "minver" : "0", "comentarii" : "...", "linii modificate" : [ ] }
fisier14.cpp	591583046dfa030f98a6b507	{ "prj" : _idPrj1, "dezv" : _idDev1, "MAJVER" : "0", "minver" : "1", "comentarii" : "...", "linii modificate" : ["2", "15", "43"] }

Menţionăm că fiecare înregistrare (document), indiferent de tabelul (colecţia) unde rezidă şi indiferent că este BLOB sau structură de date native sau de orice fel, se bucură de propriul identificator unic, care este, aşa cum s-a menţionat mai sus, un string de 24 de caractere, generat aleatoriu, automat, imediat la inserare, de către Serverul MongoDB.

5. Driverul MongoDB C#

Pentru interacţiunea între codul aplicaţiei dezvoltate şi un sistem de gestiune a bazelor de date se foloseşte un echipament de programare (în fapt, o interfaţă API) numit driver, acesta conţinând funcţii integrate pentru: (a) conectarea la serverul de baze de date şi (b) vehicularea de date, pe care aplicaţia le trimite către - sau le regăseşte din baza de date.

Într-o formulă de lucru de tip client-server, aplicaţia client nu va face altceva decât să trimită comenzi către un ServiciuWeb (evident, aflat la distanţă) şi să proceseze răspunsul primit (care, de exemplu, poate fi o confirmare de succes sau o informare de eşec – în cazul operaţiilor de scriere pe baza de date, sau un şir de numere/texte/structuri de date etc. – în cazul operaţiunilor de citire din baza de date), neavând nevoie să utilizeze driverul MongoDB; dimpotrivă, Serviciul Web va fi elementul funcţional care va utiliza driverul MongoDB, pe maşina server.

Driverul este un ansamblu compus din două biblioteci de tip C#.net, şi anume: **MongoDB.Bson.dll** şi **MongoDB.Driver.dll** care conţin clasele cu metodele cărora se realizează operaţiunile de conectare şi dialog cu baza.

Se redau mai jos câteva exemple strict ilustrative, cu explicaţii alăturate:

```
// conectare
MongoClient client = new MongoClient("mongodb://localhost:27017");
// referinţă către conectorul la sistemul de gestiune
MongoServer server = client.GetServer();
// referinţă către conectorul la  entitate bază date
MongoDatabase Baza = server.GetDatabase("COMPANIA-SOFTWARE");
// referinţă la  // lista de tabele
IEnumerator<string> tabele = Baza.GetCollectionNames().GetEnumerator();
// referinţă la un tabel
MongoCollection<object> collection = Baza.GetCollection<object>("Proiecte");
// referinţă la un fişier stocat recuperabil după ID
MongoGridFSFileInfo blobul = database.GridFS.FindOne
(Query.EQ("_id", new ObjectId("714AAABBB999852427006003")));
```

Este important a se menţiona metoda `Eval()` a clasei `Database`, care primeşte drept argument un text de tip Javascript în care se inserează comenzi specifice Mongo DB (perfect echivalent unui text de tip SQL), ca în exemplul de mai jos, unde se returnează numărul de documente (înregistrări) din colecţia (tabela) "Dezvoltatori":

```
int nr = Baza.Eval("function() { return db.Dezvoltatori.count(); }");
```

În exemplul de mai jos se va returna un text de tip listă de structuri JSON, fiecare structură conţinând toate informaţiile găzduite despre un anume fişier dezvoltat de angajatul "Ionescu".

```
string Fis = Baza.Eval
("function() { return db.Fisiere.find({dezvoltator:"Ionescu"}); }").
ToJson();
```

În SQL clasic s-ar fi returnat un recordset obţinut în urma executării comenzii:

```
SELECT * FROM 'Fisiere' WHERE dezvoltator = 'Ionescu';
```

În exemplul următor se va ilustra căutarea unui fişier după metadate; se va prezenta textul JavaScript de trimis drept argument metodei `Eval()` cu scopul de a găsi **şirul** tuturor fişierelor dezvoltate de angajatul "Ionescu" sortate descrescător după versiunea minoră:

```
function()
{
    var Blobii = db.SURSE.files;
    var conditia = {'metadata.Dezvoltator' : 'Ionescu'};
    var sirul = null; // rezultatul, de tip [] = JS Array
    sirul = Blobii.find(conditia).sort({'metadata.VersiuneMinora': -1}).toArray();

    // SAU
    // .limit(1).toArray(); // sirul care contine doar primul document

    // SAU
    // Blobii.distinct('filename', conditia).sort() // sunt excluse repetarile de
    // nume
    // distinct() intoarce un sir
    return sirul; // find() returneaza un cursor, de aceea se invoca .toArray()
}
```

Comenzile MongoDB, respectiv SQL, pentru obţinerea tuturor datelor despre toţi programatorii care au experienţă de minimum 4 ani în C++ sunt cele de mai jos:

```
db.dezvoltatori.find
(
      {
            experienta:
            { $elemMatch: { limbaj: "C++", ani: { $gt: 4 } } }
      }
)
SELECT * FROM 'Dezvoltatori' WHERE limbaj = 'C++' AND ani >= 4;
```

Concluzii

Lucrarea de faţă se concentrează pe prezentarea câtorva consideraţii asupra bazelor de date ne-relaţionale. Principala trăsătură a bazelor de date NoSQL este aceea că permit stocarea datelor în mod ne-structurat (NU sub formă de tabel clasic-schemă rigidă, ci sub formă arborescentă – schema fiind inexistentă, ceea ce face posibilă găzduirea dinamică a oricărei configurări imaginabile, cu condiţia respectării sintaxei JSON), ceea ce sporeşte considerabil eficienţa accesării lor. Se pot memora date în orice combinaţie posibilă (texte, numere întregi, fişiere, colecţii de entităţi etc. sau orice fel de grupare imaginabilă a acestora). În prezent, există multe companii care au dezvoltat propriile baze de date NoSQL.

Principalele trăsături funcţionale ale bazelor NoSQL sunt lipsa schemei, simplitatea anatomică - ceea ce permite executarea de interogări rapide, posibilitatea de a adăuga/edita/şterge dinamic noi atribute (câmpuri) la înregistrările existente, posibilitatea de a partaja încărcarea pe mai multe servere (adică distribuirea "efortului" de calcul), posibilitatea de a replica datele pe mai multe servere şi accesare simultană posibil problematică faţă de modelul relaţional (negarantarea tranzacţiilor).

În ceea ce priveşte partea de dezvoltare software, bazele NoSQL sunt open-source şi se bucură de colecţii generoase de funcţii API pentru dialogul cu aplicaţii scrise în diverse limbaje. În această ordine de idei, lucrarea face o prezentare sugestivă a utilizării Driverului MongoDB C#, ansamblu de două librării de tip **.Net** care conţin clasele cu metodele cărora se realizează operaţiunile de conectare şi dialog cu o bază de date. Pe exemple concrete şi simple, este ilustrată totodată echivalenţa dintre sintaxa clasică SQL şi sintaxa NoSQL de sorginte Javascript.

Contribuţia autorului constă în (a) - prezentarea unui nou concept informatic şi (b) – utilizarea acestuia într-un proiect concret de cercetare aplicată.

BIBLIOGRAFIE

1. *** - "C# Driver" - 2018; https://docs.mongodb.com/getting-started;
2. *** - "Data storage supply and demand worldwide, from 2009 to 2020 (in exabytes)"; https://www.statista.com/statistics/751749/worldwide-data-storage-capacity-and-demand – 2018;
3. *** - "NoSQL Databases" - 2018; http://nosql-database.org/;
4. Alexandru, A., Coardoş, D., Nicolau, D. - "A Model For Future Internet Of Things-Based Remote Monitoring Of Chronic Diseases"; Proceedings of the IE 2018 International Conference - Iaşi, 2018;
5. Andersson, Erik & Berggren, Zacharias - "A Comparison Between MongoDB and MySQL Document Store Considering Performance"; http://www.diva-portal.org/smash/get/diva2:1161166/ FULLTEXT01.pdf - 2017;

6. Băjenaru, L., Marinescu, I. A., Tomescu, M., Savu, D. - "Biblioteca Naţională de Programe: O nouă abordare în managementul produselor software" - RRIA, vol. 27, nr. 4, decembrie 2017; https://rria.ici.ro/rria-vol-27-nr-4-2017;
7. Nicolau, D. & colab. - ICI Bucureşti - "Proiect PN 16 09 03 02 - Servicii Cloud cu suport baze de date NoSQL destinate dezvoltării de proiecte Open Source; Faza 1 - Dec 2016";
8. Nicolau, D. & colab. - ICI Bucureşti - "Proiect PN 16 09 03 02 - Servicii Cloud cu suport baze de date NoSQL destinate dezvoltării de proiecte Open Source; Faza 2 - Iun 2017";
9. Shalom, N. - "The Common Principles Behind The NoSQL Alternatives", Dec 2009 http://natishalom.typepad.com/nati_shaloms_blog/2009/12/the-common-principlesbehind-the-nosql-alternatives.html;
10. Welsh, M., Culler, D., Brewer, E. - "An architecture for well conditioned, scalable internet services" - 2001; http://www.sosp.org/2001/papers/welsh.pdf.

Dragoş NICOLAU este absolvent al Facultăţii de Electrotehnică din cadrul Universității *Politehnica* Bucureşti, din anul 1991. În prezent este cercetător științific III în cadrul ICI, Bucureşti. Este specializat în dezvoltarea de aplicaţii Web, desktop şi reţea. Este puternic pasionat de zonele puţin explorate ale programării orientate pe obiect. Drept arii de interes, Dragoș Nicolau mai menționează: securizarea rețelelor, securizarea codurilor javascript şi fizica semiconductoarelor. Domnului Dragoș Nicolau îi place să studieze şi să implementeze aplicaţii software bazate pe fire de execuție, algoritmi de criptare/compresie, analiza de imagine şi comunicații rețea. A publicat peste 30 de articole în ţară şi străinătate. Între anii 1997-2002 a desfăşurat activitate didactică la Facultatea de Electrotehnică din UPB. Dragoș Nicolau este vorbitor de italiană, franceză şi engleză.

Dragoș NICOLAU is a graduate of the Faculty of Electrical Engineering at the Politehnica University of Bucharest, since 1991. Currently he holds the position of scientific researcher III at ICI, Bucharest. He has skills in developing web, desktop and network applications. He is strongly passionate about the less-explored areas of object-oriented programming. As a field of interest, Dragoș Nicolau also mentions: securing networks, securing javascript codes and semiconductor physics. Mr. Nicolau loves to study and deploy software based on execution threads, encryption / compression algorithms, image analysis, and network communications. He has published more than 30 articles in the country and abroad. Between 1997-2002 he performed Academic teaching activity at the Faculty of Electrical Engineering of UPB. Dragoș Nicolau is speaking Italian, French and English.