CZU 004.655

IMPLEMENTAREA CONCEPTULUI *GraphQL* LA GESTIUNEA DATELOR STUDENȚILOR

Valeria CABAC, *studentă, Facultatea de Științe Reale, Economice și ale Mediului, Universitatea de Stat „Alecu Russo" din Bălți*
Conducător științific: **Vitalie ȚICĂU,** *asist. univ.*

Abstract: *This article takes the method using ORM model and GraphQL system to declaring and querying objects. The relational type of databases useful for storage data of different types. On the one hand, the process of storage is fast when the type of data is primitive. On the other hand, the objects involve a complex construction, from several elements, which are of a different primitive type. Arises a conflict situation: the relational databases process simple data, and the objects sum up mode data in themselves, which may have different types, but using the relational type for objects is too expensive. To solve this problem, the ORM concept was developed, which allows saving lines of code, which leads to a significant increase in the concrete, fast and concise querying databases, when it is about the storage of objects. An example of a structure that uses ORM is the GraphQL system, which is used to declare and query data structures, most exactly – objects.*

Keywords: *databases, SQL and NoSQL query, object data type, ORM model, GraphQL system.*

Aplicarea bazelor de date la gestiunea datelor studenților

Bazele de date prezintă modalitatea unică de stocare a unor informații sau date.

În prezent, aproape că nu există aplicații care nu ar avea legătură cu baze de date, deoarece orice tip de aplicații, fie desktop, fie web, au nevoie de un mecanism de stocare, de prelucrare și de manipulare într-un mod facil a informațiilor specifice. Aceste tipuri de mecanisme sunt solicitate în diferite domenii: economie (ca de exemplu, păstrarea datelor clienților unei bănci), pedagogie (păstrarea și manipularea datelor despre elevi sau studenți ai unor instituții), rețele de socializare (păstrarea informațiilor utilizatorilor unei rețele de socializare) etc. Actualitatea mecanismelor izbucnește din cauza cerințelor pe piață: există o mulțime de date, toate fiind necesar de a fi prelucrate, păstrate, gestionate rapid și sigur. Pe hârtie, procesele date ar lua prea mult timp, de aceea este evidentă actualitatea mecanismelor automate.

Mecanismul de lucru: în general, toate bazele de date sunt manipulate de către un mecanism unic, numit Sistemul de Gestionare a Bazelor de Date (abreviat

SGBD). El este format din toate programele, interfețele și procesele care presupun crearea, inserarea, schimbarea și distrugerea bazelor de date [1].

Sunt mai multe tipuri de baze de date, fiecare având modul de reprezentare și gestionare diferit. Primul tip este cel *relațional*, care presupune că fiecare bază de dată este reprezentată sub formă de tabele. Aceste tabele conțin câmpuri care permit sortarea informației și datelor propriu-zise. Ca de exemplu, tabelul sub denumirea de „Studenții grupei IS11Z" conține câmpurile: *Numărul de Ordine*, *Numele Studentului*, *Prenumele Studentului*, *Data Nașterii*, *Tutore*. Fiecare înregistrare posedă un cod unic, numit *cheie*. În exemplul dat, câmpul care presupune această cheie este *Numărul de Ordine*. Acest câmp are o importanță enormă, întrucât permite stabilirea unor relații între mai multe tabele. În cele mai dese exemple din practică, coloana care presupune stocarea elementelor-cheie se numește *ID*. În tabelul 1 este prezentată structurarea câmpurilor indicate într-un tabel de date de tip relațional.

Tabelul 1. *Exemplu grafic al unui tabel de date*

Studenții din grupa GR11Z				
Nr.	*Numele Studentului*	*Prenumele Studentului*	*Data Nașterii*	*Tutore*
1.	Nume	Prenume	15.06.2001	
2.				
3.				
4.				
5.				
6.				

Din denumire se înțelege că între câmpurile unor tabele există o *relație* anumită. Adică, pot exista câteva tabele, care presupun o legătură (sau mai multe) între valorile acestora. Fie că este dat tabelul *Profesorilor* unei Facultății, în care sunt indicați profesorii și obiectele, care le predau. Acest tabel, de asemenea, trebuie să conțină un câmp-cheie, fie *Numărul de Ordine* (tabelul 2).

Tabelul 2. *Tabel de date pentru profesori*

Profesori			
Nr.	*Nume Prenume*	*Obiectul predat*	*Facultatea*

Baza de date a studenților din grupa GR11Z va avea, în acest caz, o legătură cu tabelul *Profesorilor*, astfel se stabilește o relație între câmpul *Tutore* al primului tabel și câmpul *Nr* a tabelului al doilea. Acesta este un tip de relație, sub denumirea de *unu la unu* sau *one to one*: un câmp are legătură cu alt câmp, prestabilit.

În tabelul 3 este prezentat un alt tabel al ierarhiei – tabelul *Permiselor* la Bibliotecă al studenților. Acest tabel conține câmpurile: *Numărul de Ordine*, *Numele Prenumele*, *Grupa*, *Facultatea*. Astfel, acest tabel are legătură cu două alte tabele concomitent – tabelul *Grupei* și tabelul *Profesorilor* facultăților. Acest tip de legătură se numește *unul la mai multe*, sau *one to many*.

Tabelul 3. *Tabel de date pentru permise la bibliotecă*

Permisele la Bibliotecă			
Nr.	*Numele Prenumele*	*Grupa*	*Facultatea*

Pentru accesarea şi gestiunea bazelor de date se foloseşte limbajul standard de interogare SQL. În acest limbaj se generează interogări, adică se selectează datele după criteriile specifice. Interogările sunt generate prin instrucțiuni, care au efect persistent asupra datelor sau structurilor de date, sau pot controla tranzacțiile, conexiunile pe parcursul îndeplinirii programului.

SQL conține instrucțiuni cu o mulțime variată de elemente: clauze (componente ale instrucțiunilor şi interogărilor), expresii (cu efect de producere a valorilor scalare sau tabelare), predicate (specifică condițiile evaluate de SQL conform logicii ternare sau logicii booleene în scopul limitării efectelor instrucțiunilor sau pentru a influența cursul programului).

Tipurile de date în SQL sunt în funcție de tipul informației din câmpul tabelului. De exemplu, în câmpul care presupune ID-ul unui element, este clar că va fi de tip INTEGER (sau numit şi SMALLINT), adică va fi un număr întreg. La fel, câmpul care presupune numele/prenumele va fi numaidecât de tip CHARACTER (sau CHAR), adică şir de caractere. Alte tipuri de date folosite sunt: REAL (număr real), NUMERIC (număr zecimal cu precizia cifrelor din partea întreagă şi numărul de zecimale), DATE (data zilei), TIME (ora) [1].

Drept exemplu, folosind un dialect din familia SQL, numit MySQL, se poate crea o bază de date a grupei GR11Z.

CREATE TABLE numele_tabelului(câmp tip, câmp tip, câmp tip primary key);
CREATE TABLE GR11Z (int ID primay key, char Nume, char Prenume,
date DataNasterii, char Adresa, char Email, char Grupa);
INSERT INTO GR11Z (ID, Nume, Prenume, DataNasterii, Char, Email, Grupa)
VALUES(1, Cabac, Valeria, 15.06.2001,
Balti strada Bulgara 1/10, cabacv15@gmail.com, gr11z);

Stocarea datelor în limbajul SQL poate fi efectual nu doar prin scrierea instrucțiunilor direct, ci şi prin folosirea a unor interfețe (ca de exemplu, MySQL conține şi Workbench, care este un instrument vizual pentru arhitectura bazelor de date) [2].

Tipul de date obiect

În general, tipul de baze relațional este unul foarte util la capitolul stocării datelor de diferite tipuri. După cum s-a menționat mai sus, procesul de stocarea este rapid, atât timp cât se lucrează cu datele strict de tip primitiv. Dar, în ziua de azi, se foloseşte, în bună parte, o nouă paradigmă în programare, unde totul este estimat ca un obiect, ceea ce face logica programării mai aproape de lumea fizică. Obiectele presupun o construcție complexă, creată din mai multe elemente, care sunt de tip primitiv diferit. Însă apare o situație conflictuală: bazele relaționale procesează date simple, iar obiectele însumează în sine câteva date, care pot avea tip diferit. Folo-

sirea tipului relațional pentru obiecte este prea costisitoare, deoarece este nevoie de creat mai multe baze de date, pentru a evita un break semantic. Evitarea acestui fenomen ar impune programatorul să creeze linii de cod, softuri, programe speciale, care ar putea să prelucreze datele ca niște structuri de tip orientate pe obiect, cât și să le păstreze în tabele relaționale. Consecințele sunt că pentru afișarea informației totale, va fi nevoie de a formata mai multe interogări, ceea ce necesită mai mult timp de codificare și de executare. Plus la aceasta, tranzicția datelor va fi una de lungă durată și costisitoare din punct de vedere al memoriei.

Pe parcurs au fost proiectate mai multe variante de mecanisme, care ar putea înșătura necesitatea transformării obiectelor pentru a le stoca fără erori în baze de date relaționale. Unele din variante au fost biblioteci speciale de clase, care pot realiza, în mod automat, convertirea lor în tipuri primitive de date. De exemplu, având lista de tabele în baze de date și obiecte din program, convertirea interogărilor SQL necesare se vor genera și executa automat, convertind rezultatul în obiect. Teoretic, având lista de obiecte (Facultăți, Profesori, Grupe, etc.) și lista de tabele (Facultățile Universității, Profesorii Facultății, Grupe), softul, creat pentru conversie, prelucrează obiectele în mod implicit și le salvează automat în baza de date relaționale. În practică, totuși, totul este un pic mai complicat. Tranzițiile pot fi lente și ineficiente, întrucât generarea interogărilor SQL poate fi neprevăzută, ceea ce duce la utilizarea memoriei într-o cantitate mai mare, decât aceleași programe, scrise „manual”.

Variante pentru soluționare au fost mai multe, printre care și protocolul SOAP (Simple Object Access Protocol), bazat pe XML. Însă, din cauza conflictelor dintre cod XML și alte limbaje, cât și reguli stricte de utilizare, ca un răspuns la aceasta vine REST (Representational State Transfer), care folosește doar abordarea URL-urilor. Pentru răspunsuri, acesta generează formate JavaScript Object Notation (JSON), Command Separated Value (CSV) și Really Simple Syndication (RSS). Ideea este de a obține rezultatul de care este nevoie într-o formă mult mai concisă.

Modelul obiect relațional

Totuși, complexitatea programelor crește, astfel încât chiar și REST devine mai dificil de folosit. REST, în cea mai mare măsură, este conceput pentru aplicații network, iar rezultatul interogării acestui API este un dataset complet. Acest fapt trezește conflicte când se cere informații din două baze de date, complet diferite. Astfel, atunci când interogările sunt mai complexe, iar protocoalele sunt diferite, programatorul întâlnește problema deja cunoscută: tranziția lentă, numărul enorm de cod și utilizarea inadecvată a memoriei.

Pentru a soluționa această problemă a fost elaborat un concept, care permite manevrarea datelor din obiecte [3]. Misiunea generală a *Conceptului Obiect - Relațional* (abreviat, ORM) este economisirea rândurilor de cod, ceea ce duce la mărirea semnificativă a vitezei de execuție a programului. Plus la aceasta, o mare parte a implementărilor acestui model permite definirea, schimbarea, descrierea codului de interogare SQL, care va fi utilizat pentru acțiuni ca salvarea bazei, încărcarea, căutarea etc., la necesitate cu un obiect constant.

Astfel, modelul ORM permite interogarea concretă, rapidă și concisă a bazelor de date, când se discută despre păstrarea obiectelor.

Sistemul GraphQL

Un exemplu de structură de acest tip este GraphQL. Ca o definiție generală, GraphQL este o sintaxă, care se utilizează pentru declararea și interogarea structurilor de date, adică a obiectelor. Întocmirea unei declarații sau interogări este una simplă. De exemplu:

```
University {
Facultati {
        NumePrenume
        Obiect
}
}
```

În secvența aceasta este indicat concret ce date solicităm din întreaga bază, folosind o structură, din punct de vedere grafic asemănătoare cu un graf, iar răspunsul va fi de tip JSON. Deci, din secvență este clar că solicităm doar două câmpuri (*NumePrenume*, *Obiect*) dintr-un tabel (*Facultati*). Această interogare se va executa rapid, întrucât solicitarea este de minim elemente, ceea ce duce la tranziția a minim de elemente. Dar și forma dată este una mai ușor de scris, deci mai comodă.

Teoria Grafurilor a pornit de la amuzamente matematice, ca apoi să devină obiecte de studii și principii, pe care se bazează multe arhitecturi și algoritme în informatică. Definiția grafului presupune un grup de obiecte, între care există legături. Obiectele se mai numesc vârfuri sau noduri, legăturile sunt numite muchii [4].

Principiile GraphQL sunt următoarele [5, p. 18]:

- *Modelul ierarhic* – câmpurile sunt încorporate în alte câmpuri, iar forma interogării este identică cu forma datelor returnare.
- *Orientarea către produs* – GraphQL presupune satisfacerea condițiilor datelor clienților, precum și a limbajului, timpului de rulare așteptat.
- *Tipologia strictă* – Serverul GraphQL este acceptat de sistemul GraphQL, astfel încât grafic, orice punct al schemei de date are tipul său propriu, în conformitate cu care va fi prelucrat la interogare.
- *Interogări, determinate de client* – Serverul GraphQL permite manevrarea interogărilor după cerințele concrete ale utilizatorului.
- *Introspectivă* – limbajul GraphQL poate interoga sistemul unui Server GraphQL.

Revenind la definiție, se accentuează faptul că GraphQL este o sintaxă, o specificație. Legătura cu baza de date se face în funcție de limbajul pe care îl folosim.

GraphQL nu este bazat pe utilizarea unei singure baze de date. Dar dacă există mai multe baze, toate fiind scrise în diferite limbaje, de exemplu SQL și NoSQL, procesul de combinare și unire a bazelor totuși are loc. Există o aplicație-client și două servere diferite, cu API (*Application Programming Interface – descrierea tuturor modurilor de interogare dintre două sau mai multe programe)* diferite. Pentru stabilirea unei legături sigure, este nevoie de a folosi anumite protocoale (*http, ssh, ws*, cli), și, în cazul când protocoalele serverelor diferă, este nevoie de a folosi un API Gateway. Un API Gateway este un instrument de utilizare a API-ului, când este vorba de crearea unei arhitecturi multitentative. Aceasta prezintă un

microserver, care permite, în cel mai optimizat mod, folosirea resurselor şi optimizarea cheltuielilor în interogări cu mai mulți clienți. De fapt, GraphQL este un exemplu de API Gateway, deoarece el este ca un strat adiacent între aplicație-client şi servere. Transportarea rezultatelor interogărilor pe traseul client-server se execută cu ajutorul oricărui protocol, în funcție de cel utilizat de server în felul următor: se cere o resursă anumită de la GraphQL server în primul rând, acest sever analizează interogarea, parcurge graful în mod recursiv şi execută pentru fiecare câmp interogarea dată; apoi, când toate datele sunt primite de la serverele bazelor de date GraphQL-server întoarce rezultatul aplicației-client.

Pe lângă interogări, GraphQL poate fi folosit la adăugarea unor elemente în bază de date, poate fi ca un listener, care permite observarea schimbărilor în timp real (folosit la primirea listei de persoane, care au reacționat la publicații în rețele de socializare) etc.

Bibliografie:

1. БЬЮЛИ, А. *Изучаем SQL*. Санкт-Петербург: Симбо, 2007. 301 с. ISBN 978-5-932-860-519.
2. *MySQL Workbench* [online] [citat 21.03.2021]. Disponibil: https://www.mysql.com/products/workbench
3. *Mapping Objects to Relational Databases* [online] [citat 21.03.2021]. Disponibil: http://www.agiledata.org/essays/mappingObjects.html
4. NĂDĂBAN, S., ŞANDRU, A. *Algoritmica grafurilor*. Timişoara: Mirton, 2007. 265 p.
5. БЭНКС, А., ПОРСЕЛЛО, Е. *GraphQL. Язык запросоов для современных веб-приложений*: Санкт-Петербург, 2019, 240 с. ISBN 978-5-4461-1143-5.