**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**UNIVERSITATEA DE STAT „ALECU RUSSO” DIN BĂLȚI**

**FACULTATEA DE ȘTIINȚE REALE, ECONOMICE ȘI ALE MEDIULUI**

**CATEDRA DE MATEMATICĂ ȘI INFORMATICĂ**

**PRIORITATEA ORM FAȚĂ DE BAZELE DE TIP RELAȚIONAL DE DATE**

**REFERAT LA CURSUL INFORMATICA GENERALĂ**

**Autor:**

Studenta grupei IS11Z

**Valeria CABAC**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Conducător științific:**

**Olesea SKUTNIȚKI**

magistru, asist.univ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**BĂLȚI, 2021**

CUPRINS

[CUPRINS 2](#_Toc71782123)

[INTRODUCERE 3](#_Toc71782124)

[1 Aplicarea bazelor de date la gestiunea datelor 4](#_Toc71782125)

[1.1 Mecanismul de lucru 4](#_Toc71782126)

[2. Tipul relațional de baze de date 5](#_Toc71782127)

[2.1 Tipuri de relații 5](#_Toc71782128)

[3.SQL – limbajul standart de interogare 7](#_Toc71782129)

[4. Modelul Relațional 8](#_Toc71782130)

[4.1 Teoria Grafurilor 8](#_Toc71782131)

[4.2 Modelul GraphQL 8](#_Toc71782132)

[BIBLIOGRAFIE 10](#_Toc71782133)

INTRODUCERE

*Bazele de date* reprezintă o colecție bine organizată de diferite tipuri de informații sau de date structurate, stocate electronic pe un suport fizic. [1] Există mai multe tipuri de baze de date, pentru a stoca datele de diferit tip într-un mod mai facil.

Tipul de baze relațional este foarte util pentru stocarea datelor de diferite tipuri. Procesul de stocarea este rapid, însă doar atunci când se lucrează cu datele strict de tip primitiv. Tip primitiv presupune tipul integer, char, double, float și chiar și string. Spre exemplu, pentru stocarea informației, privind un an anumit, presupune folosirea tipului integer. Sau, stocarea informației despre numele unei persone, presupune tipul de date string.

Dar, pe lângă datele de tip simplu, există și așa numitele *obiecte,* o noțiune relativ nouă, însă foarte actuală în informatică*.* Obiectele presupun o construcție complexă, creată din mai multe elemente, care sunt de tip primitiv diferit. Obiectele sunt menite pentru a facilita modelarea entităților și logicii complexe din lumea reală, iar reutilizarea obiectelor permit dezvoltarea aplicațiilor de baze de date mult mai rapid și mai eficient. Obiectele însumează în sine nu doar tipuri de date, dar și metode aplicate asupra entităților consistente [2].

Apare o situație de conflict: bazele relaționale procesează date simple, iar obiectele însumează în sine câteva date, care pot avea tip diferit, iar folosirea tipului relațional pentru obiecte este prea costisitoare. Pentru a soluționa această problemă a fost elaborat conceptul ORM, care permite economisirea rândurilor de cod, ceea ce duce la mărirea semnificativă a vitezei de execuție a programului. De asemenea, modelul ORM permite interogarea concretă, rapidă și concisă a bazelor de date, când se discută despre păstrarea obiectelor.

# 

1 Aplicarea bazelor de date la gestiunea datelor

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.

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

2. Tipul relațional de baze de date

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[3].

Sunt mai multe tipuri de baze de date, fiecare având modul său 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.

2.1 Tipuri de relații

Ca de exemplu, fie tabelul sub denumirea de „Studenții grupei GR11Z”, care conține câmpurile: *Numărul de Ordine*, *Numele Studentului*, *Prenumele Studentului*, *Data Nașterii*, *Tutore* (fig.1).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Studenții grupei GR11Z | | | | |
| Nr. | Numele Studentului | Prenumele Studentului | Data Nașterii | Tutore |
| 1. |  |  |  |  |
| 2. |  |  |  |  |

(fig.1. *Tabelul studenților grupei GR11Z*)

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 semnificativă, întrucât anume el permite stabilirea unor relații între alte tabele. În cele mai dese exemple din practică, coloana care presupune stocarea elementelor-cheie se numește *ID*. În tabelul din figura 1 este prezentată structurarea câmpurilor indicate într-un tabel de date de tip relațional.

Însăși din denumire se înțelege, că între câmpurile unor tabele există o *relație* anumită. Adică, pot exista câteva tabele, între care există 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* (fig. 2).

|  |  |  |  |
| --- | --- | --- | --- |
| Profesori | | | |
| Nr. | Nume Prenume | Obiectul Predat | Grupa |
|  |  |  |  |
|  |  |  |  |

(fig.2 *Tabelul Profesorilor*)

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 figura 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 *Facultăților*. Acest tip de legătură se numește *unul la mai multe*, sau *one to many*.

|  |  |  |  |
| --- | --- | --- | --- |
| Permise | | | |
| Nr. | Numele Prenumele | Grupa | Facultatea |
|  |  |  |  |
|  |  |  |  |

(fig. 3 *Tabelul Permiselor*)

3.SQL – limbajul standart de interogare

Pentru accesarea și gestiunea bazelor de date se folosește 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 și 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 dependență 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 și SMALLINT), adică va fi un număr întreg. La fel, câmpul care presupune numele/prenumele va fi numaidecât de tip CHARACTER (sau CHAR), adică șir de caractere. Alte tipuri de date folosite sunt: REAL (număr real), NUMERIC (număr zecimal cu precizia cifrelor din partea întreagă și numărul de zecimale), DATE (data zilei), TIME (ora) [].

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*](mailto:cabacv15@gmail.com)*, gr11z);*

Stocarea datelor în limbajul SQL poate fi efectual nu doar prin scrierea instrucțiunilor direct, ci și prin folosirea a unor interfețe (ca de exemplu, MySQL conține și Workbench, care este un instrument vizual pentru arhitectura bazelor de date) [4].

4. Modelul 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, cât și 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 [5]. 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.

4.1 Teoria Grafurilor

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 [6]

4.2 Modelul 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.

În secvența de cod 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. Interogările 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.

Principiile GraphQL sunt următoarele [7, 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 condițiile datelor ale clienților, precum și a limbajului, timpului de rulare aștept.

*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 dependență 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 merge vorba de crearea unei arhitecturi multitentative. Aceasta prezintă un microserver, care permite, în cel mai optimizat mod, folosirea resurselor și 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 și servere. Transportarea rezultatelor interogărilor pe traseul client-server se execută cu ajutorul oricărui protocol, în dependență 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, și 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.

Concluzie

În ziua de azi, bazele de date, prezintă, în principiu, cel mai important mecanism în orice domeniu, întrucât datele pe zi ce trece cresc, iar manipularea lor poate avea o importanță decesivă și iminentă. Cantitatea datelor crește în progresie geometrică, cât și complexitatea lor. Menirea inovațiilor în baze de date, accentuând modelul ORM, poate chiar și viitoarele modele care vor mai apărea, este de ușurarea, inovarea și oferirea programării și proiectării bazelor o vizualitate mul mai aproape de lumea reală, cât și mult mai aproape de paradigmele de programare.

Totuși, nu ar fi corect de discriminat tipul relațional, cât și de amplificat importanța modelului ORM. Fiecare model de bază de dată este stric menit pentru anumite cazuri, probleme informatice, etc, cel relațional prelucrând cu excelență enormă datele de tip primitiv, iar cele complexe sunt manipulate de ORM.

În concluzie, se mai adaugă că mecanismele sunt in permanentă schimbare, tipurile sunt în permanentă schimbare, fiecare programator în parte este capabil să-și aleagă metoda sa.

BIBLIOGRAFIE

1. *Ce este o bază de date?* [online]. Disponibil în Internet pe adresa:

<https://www.oracle.com/ro/database/what-is-database/> (accesat 07.05.2021);

1. *Database Application Developer’s Guide – Object-Relational Features* [online]. Disponibil în Internet pe adresa:

<https://docs.oracle.com/cd/B19306_01/appdev.102/b14260/adobjint.htm> (accesat 09.05.2021);

1. *What is database?*[online]. Disponibil în Internet pe adresa:

<https://www.oracle.com/database/what-is-database/> (accesat 20.04.2021);

1. БЬЮЛИ, А. *Изучаем SQL*. Санкт-Петербург, Симбо, 2007, 301 с. ISBN 978-5-932-860-519.
2. *Mapping Objects to Relational Databases*. [online] Disponibil pe Internet (21.03.2021).

<http://www.agiledata.org/essays/mappingObjects.html>

1. NĂDĂBAN, S.; ȘANDRU, A. Algoritmica grafurilor Timișoara, Mirton, 2007, 265p
2. БЭНКС А.; ПОРСЕЛЛО Е. *GraphQL. Язык запросоов для современных веб-приложений*,  [Питер](https://www.labirint.ru/pubhouse/104/), 2019, 240 с. ISBN 978-5-4461-1143-5.