Universitatea „Petrol – Gaze” din Ploiești

**Informații de interes public zonal**

Realizat de: Coșereanu Emanuel, INFO an 2, grupa 40322

Cuprins

[Introducere 3](#_Toc199533077)

[Normalizare 4](#_Toc199533078)

[Baze de date tranzacționale 6](#_Toc199533079)

[Tabele 7](#_Toc199533080)

[Operații 14](#_Toc199533081)

[Reuniune 14](#_Toc199533082)

[Diferență 15](#_Toc199533083)

[Selecție 15](#_Toc199533084)

[Proiecție 16](#_Toc199533085)

[Joncțiune „simplă” 17](#_Toc199533086)

[Joncțiune „compusă” 1 17](#_Toc199533087)

[Joncțiune „compusă” 2 18](#_Toc199533088)

[Concluzie 19](#_Toc199533089)

# Introducere

Pentru rezolvarea acestei teme am făcut o copie a bazei de date pentru a lucra, din dorința de a avea mereu și baza de date veche pentru a le putea compara pe cele două după finalizarea proiectului.

Imediat cum m-am apucat de lucru am observat că tabela **chitante** nu are ce căuta în baza mea de date, deoarece nu are o legătură strânsă cu tema aplicației. Practic, eu doream ca utilizatorul să poată vedea toate bonurile fiscale (care din nou, nu au treabă cu chitanțele) de la toate localurile la care a fost, doar că există o problemă de politică, în sensul că fiecare local ar trebui să își dea acordul pentru a putea avea datele acestora, iar ei la rândul lor, ar trebui să folosească aplicația mea, ceea ce e aproape imposibil. Din acest motiv am renunțat la tabela de mai sus.

Mai jos voi lăsa o poză cu baza de date inițială, nenormalizată (Fig. 1.1), urmând ca după ce voi explica cum am gândit procesul de normalizare, să las o poză și cu cea normalizată.

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 1.1** Baza de date nenormalizată

# Normalizare

Procesul de normalizare a început prin verificarea dacă tabelele se află în FN1 și am găsit imediat câteva probleme.

În tabelele **localuri** și **firme\_livrare** aveam câmpul **program** care memora într-un fel o listă/mulțime, deoarece înregistrările din acest câmp arătau în felul următor: „Luni-Vineri: 9:00-23:00, Sâmbătă-Duminică: 10:00-22:00”, ceea ce ar fi creat o problemă enormă dacă aș fi dorit să modific programul. Din acest motiv am creat o altă tabelă denumită **programe**, care are următoarea structură:

programe(id\_program int primary key auto\_increment, ora\_inceput varchar(5), ora\_final varchar(5), zi varchar(12), id\_local int, id\_firma int, foreign key(id\_local) references localuri(id\_local) on delete cascade, foreign key(id\_firma) references firme\_livrare(id\_firma) on delete cascade);

După crearea acestei tabele am inserat în ea toate datele referitoare la program care se aflau în cele două tabele, având grija să păstrez exact aceleași orare, urmând să șterg câmpul **program**.

În tabela **firme\_livrare** mai aveam și câmpul **zone\_acoperire** ce memora valori precum „Ploiești și împrejurimi”, motiv pentru care am creat o nouă tabelă denumită **zone\_acoperire**, pentru a memora ce zone vizează fiecare firmă de livrare.

create table zone\_acoperire(id\_zona int primary key auto\_increment, Ploiesti boolean, imprejurimi\_de\_10\_km boolean, id\_firma int, foreign key(id\_firma) references firme\_livrare(id\_firma) on delete cascade);

Totuși, după crearea acestei tabele mi-am dat seama că în componența ei nu are ce căuta câmpul **Ploiești**, motiv pentru care mi-am creat și o tabelă **orase** în care memorez numele fiecărui oraș, iar tabela **zone\_acoperire** a fost modificată astfel încât să fie legată de noua tabelă creată, așa cum se poate vedea mai jos (Fig. 2.1).

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 2.1** Tabelele **orase** și **zone\_acoperire**

Tabela **orase** a fost folosită și pentru a modifica tabela **adrese** astfel încât să fie în FN3, motiv pentru care apare câmpul **judet** în componența acesteia.

Am pus toate datele referitoare la zonele de acoperire din tabela **firme\_livrare** în noua tabelă creată și am șters câmpul din **firme\_livrare**.

În tabela **terase** aveam câmpul **pet\_friendly** ce memora serviciile pe care o terasă le oferă animalelor de companie, precum: „apă și crochete”, doar că din neatenție am trecut peste această problemă și mi-am dat seama destul de târziu, motiv pentru care am modificat câmpul astfel încât să pot memora în el doar valori de 0 și 1, 0 – terasa nu oferă nimic animalelor, 1 – terasa oferă apă și mâncare pentru animale.

Tot în timp ce eram în procesul de normalizare mi-am dat seama că atributul **cost\_standard** nu are ce căuta în tabela **firme\_livare**, deoarece există un cost standard de livrare pentru fiecare local, în funcție de firmă, motiv pentru care am mutat acest atribut în tabela de legătură **localuri\_firme\_livrare**, respectându-se cerințele pentru FN2, atributul fiind dependent în totalitate de **id\_local**, cât și de **id\_firma**, care deși sunt chei străine alcătuiesc cheia primară a tabelei.

alter table localuri\_firme\_livrare add column cost\_standard float;

alter table firme\_livrare drop column cost\_standard;

În tabela **adrese** (singura care mă putea ajuta în ceea ce privește FN3-ul, restul fiind prea „simple”) aveam câmpul **localitate** pe care l-am modificat în **oras**, deoarece fiecare oraș este unic la noi în țară și astfel vom avea următoarele dependențe: **cod\_postal -> oras**, **oras -> judet**, și pentru că aplicația mea vizează în principiu informații de interes public zonal dintr-un oraș, ori împrejurimile acestuia.

Tabela **adrese** arăta inițial în felul următor: **adrese(id\_adresa, strada, oras, numar, cod\_postal, judetul, id\_local)**, unde **id\_adresa** este cheie primară. Din cauza dependențelor de mai sus am creat următoarele tabele pentru a rezolva problema: **judete(id\_judet, nume\_judet)**, **orase(id\_oras, nume\_oras, id\_judet)**, **coduri\_postale(cos\_postal, id\_oras)**, urmând ca tabela **adrese** să fie modificată (să se șteargă câmpurile problematice și să rămână doar **cod\_postal**), și să arate în felul următor: **adrese(id\_adresa, strada, numar, cod\_postal, id\_local)**.

Mai jos voi lăsa comenzile folosite pentru crearea tabelelor (fără tabela **orase**, care a fost creată anterior și căreia i s-a atribuit câmpul **judet**, și constrângerea necesară), și pentru modificare tabelei **adrese**.

create table judete(id\_judet int primary key auto\_increment, nume\_judet varchar(55));

create table coduri\_postale(cod\_postal int primary key, id\_oras int, foreign key(id\_oras) references orase(id\_oras) on delete cascade);

alter table adrese drop column oras;

alter table adrese drop column judetul;

alter table adrese add foreign key(cod\_postal) references coduri\_postale(cod\_postal) on delete cascade;

În urma modificărilor de mai sus, baza de date normalizată a ajuns să arate așa cum se vede în Fig. 2.2.

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 2.2** Baza de date normalizată

# Baze de date tranzacționale

Pentru a lucra cu baze de date tranzacționale trebuie să folosesc motorul **InnoDB**. Acest lucru a fost modificat încă din semestrul trecut (Fig. 3.1), astfel încât baza mea de date actuală permite lucru cu tranzacții.

A screenshot of a computer program

AI-generated content may be incorrect.

**Fig. 3.1** Motorul schimbat în **InnoDB**

# Tabele

Pentru afișarea tabelelor am creat un site web, am realizat conexiunea la baza de date în fișierul **db.php**, iar apoi am creat câte o pagină pentru fiecare tabelă, iar unde era nevoie câte două pagini php. Spre exemplu, în tabela **program** memoram atât programul localurilor, cât și programul firmelor de livrare, motiv pentru care eu m-am gândit să creez 3 pagini pentru tabelele principale: **localuri, firme\_livrare** și tabela de legătură **localuri\_firme\_livrare**. Pentru a putea trece dintr-o pagină în cealaltă am folosit o bară de meniu. Astfel, când vizualizăm tabela **localuri**, mai jos avem o altă bară de meniu (secundară) care ne ajută să navigăm pentru a vedea celelalte tabele care au legătură cu tabela **localuri**, precum: **terase, gelaterii, cafenele, cofetării**, **restaurante** și bineînțeles **program, adrese, contacte**. Același lucru l-am făcut și pentru tabela firme\_livrare, motiv pentru care afișarea tabelei **programe** s-a realizat în două pagini php: **programe\_localuri.php** și **programe\_firme\_livrare.php**.

Totodată, dacă în tabele aveam valori de tip boolean, la afișarea acestora în site am afișat DA sau NU, în funcție de valoarea prezentă în tabelă.

Mai jos voi lăsa o listă cu tabele afișate și numărul pozei în care se găsesc:

* localuri (Fig. 4.1)
* cafenele (Fig. 4.2)
* restaurante (Fig. 4.3)
* cofetarii (Fig. 4.4)
* terase (Fig. 4.5)
* gelaterii (Fig. 4.6)
* programe (pentru tabela localuri) (Fig. 4.7)
* adrese (aici s-au folosit și tabelele orase, judete, coduri\_postale) (Fig. 4.8)
* contacte (pentru tabela localuri) (Fig. 4.9)
* firme\_livrare (Fig. 4.10)
* contacte (pentru tabela firme\_livrare) (Fig. 4.11)
* zone\_acoperire (Fig. 4.12)
* programe (pentru tabela firme\_livrare) (Fig. 4.13)
* localuri\_firme\_livrare (Fig. 4.14)

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.1** Tabela localuri

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.2** Tabela cafenele

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.3** Tabela restaurante

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.4** Tabela cofetarii

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.5** Tabela terase

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.6** Tabela gelaterii

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.7** Tabela programe (pentru tabela localuri)

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.8** Tabela adrese

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.9** Tabela contacte (pentru tabela localuri)

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.10** Tabela firme\_livrare

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.11** Tabela contacte (pentru tabela firme\_livrare)

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.12** Tabela zone\_acoperire

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.13** Tabela programe (pentru tabela firme\_livrare)

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 4.14** Tabela localuri\_firme\_livrare

# Operații

Pentru implementarea acestor operații am introdus câteva pagini cu informații relevante pentru utilizator. În momentul în care acesta dorește să vadă anumite informații specifice despre localurile din zonă, în spate se vor executa interogările necesare pentru a viza operațiile cerute.

## Reuniune

M-am gândit că unii utilizatori nu ar dori să vadă toate localurile din zonă, ci doar cafenelele și cofetăriile, deoarece acestea sunt cele mai relevante indiferent de buget, timp și vreme.

Interogarea SQL arată în felul următor:

SELECT nume FROM localuri WHERE id\_local IN (SELECT id\_local FROM cafenele) UNION DISTINCT SELECT nume FROM localuri WHERE id\_local IN (SELECT id\_local FROM cofetarii);

Pentru ca utilizatorul să poată vedea aceste localuri, atunci când este pe pagina **Localuri**, are prezentă o a treia bară de meniu, din care poate alege pagina **Cafenele si cofetarii** (Fig. 5.1).

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 5.1** Realizarea operației de reuniune în pagina **Cafenele si cofetarii**

## Diferență

Pentru realizarea diferenței i-am oferit utilizatorului posibilitatea să vizualizeze toate localurile ce nu au firmă de livrare proprie (Fig. 5.2).

Interogarea SQL arată în felul următor:

SELECT l.nume FROM localuri l WHERE NOT EXISTS ( SELECT NULL FROM localuri\_firme\_livrare lf JOIN firme\_livrare f ON lf.id\_firma = f.id\_firma WHERE lf.id\_local = l.id\_local AND f.nume = l.nume);

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 5.2** Realizarea operației de diferență în pagina **Localuri ce nu au firma de livrare proprie**

## Selecție

Operația de selecție a fost realizată prin a-i permite utilizatorului să vadă toate localurile ce acceptă animale de companie (Fig. 5.3). Interogarea SQL va fi lăsată mai jos.

SELECT nume from localuri where animale\_de\_companie = 1;

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 5.3** Realizarea operației de selecție în pagina **Localuri ce permit animale de companie**

## Proiecție

Pentru realizarea operației de proiecție i-am permis utilizatorului să vadă adresele de email a localurilor (Fig. 5.4), deoarece anumite localuri precum McDonald’s au mai multe restaurante, cu date diferite de contact, totuși cu aceeași adresă de email.

Interogarea SQL:

SELECT DISTINCT l.nume, c.mail FROM localuri l JOIN contacte c ON l.id\_local = c.id\_local;

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 5.4** Realizarea operației de proiecție în pagina **Adrese email**

## Joncțiune „simplă”

Pentru realizarea acestei operații m-am folosit de tabela de legătură dintre localuri și firmele de livrare pentru a-i permite utilizatorului să vizualizeze firmele de livrare cu care lucrează fiecare local și prețul standard pe care firmele de livrare îl au în funcție de localul cu care acestea lucrează (Fig. 5.5).

Interogarea SQL:

SELECT l.nume nume\_local, f.nume nume\_firma, lf.cost\_standard cost FROM localuri l JOIN localuri\_firme\_livrare lf ON l.id\_local = lf.id\_local JOIN firme\_livrare f ON lf.id\_firma = f.id\_firma ORDER BY l.nume;

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 5.5** Realizarea operației de joncțiune „simplă” în pagina **Localuri firme livrare**

## Joncțiune „compusă” 1

Pentru realizarea acestei interogări am folosit tabele temporare, codul va fi lăsat mai jos. Scopul acestei joncțiuni este de a-i permite utilizatorului să vadă toate restaurantele ce livrează cu firma Bolt și au costul de livrare mai mic de 17 RON (Fig. 5.6).

CREATE TEMPORARY TABLE firme\_bolt AS SELECT id\_firma FROM firme\_livrare WHERE nume = 'Bolt Food';

CREATE TEMPORARY TABLE livrari\_ieftine AS SELECT id\_local, id\_firma FROM localuri\_firme\_livrare WHERE cost\_standard < 17;

SELECT l.nume FROM localuri l JOIN restaurante r ON l.id\_local = r.id\_local JOIN livrari\_ieftine lf ON l.id\_local = lf.id\_local JOIN firme\_bolt f ON lf.id\_firma = f.id\_firma;

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 5.6** Realizarea operației de joncțiune „compusă” 1 în pagina **Restaurante ce livreaza cu Bolt Food si au cost standard < 17 RON**

## Joncțiune „compusă” 2

Pentru realizarea acestei operații i-am permis utilizatorului să vadă toate localurile ce livrează duminica (Fig. 5.7), acest lucru fiind posibil doar dacă lucrează cu o firmă de livrare ce livrează duminica, ori firma lor de livrare livrează duminica.

A screenshot of a computer

AI-generated content may be incorrect.

**Fig. 5.7** Realizarea operației de joncțiune „compusă” 2 în pagina **Localuri ce livreaza duminica**

Interogarea SQL:

SELECT DISTINCT l.nume FROM (SELECT id\_local FROM programe WHERE zi = 'Duminica') pl JOIN localuri l ON l.id\_local = pl.id\_local JOIN localuri\_firme\_livrare lf ON l.id\_local = lf.id\_local JOIN (SELECT id\_firma FROM programe WHERE zi = 'Duminica') pf ON lf.id\_firma = pf.id\_firma;

# Concluzie

După realizarea acestui proiect, și bineînțeles a celui de pe semestrul 1, mi-am dat seama că nu am gândit foarte bine baza de date, deoarece am multe câmpuri de tip boolean ce fac ca tabelele mele să fie foarte monotone, nu am chei compuse (doar una), motiv pentru care normalizarea a fost foarte dificilă pentru că nu înțelegeam ce ar trebui să modific.

În ansamblu tema a fost foarte interesantă. Am învățat lucruri noi atât despre baze de date cât și despre restaurante, cofetării și cafenele. Mi-aș fi dorit să fiu mult mai organizat de la început și baza de date să fie creată mai profesionist.