Baze de date

Curs 6 – Normalizarea bazelor de date

Sorina Preduț
sorina.predut@my.fmi.unibuc.ro
Universitatea din București

Proiectarea bazelor de date - recap.

- Metodele curente de proiectare a BD sunt divizate în 3 etape:
  1. crearea schemei conceptuale (curs3),
  2. crearea design-ului logic al bazei de date (curs3) şi
  3. crearea design-ului fizic al bazei de date (cursul curent).

Proiectarea bazelor de date - recap.

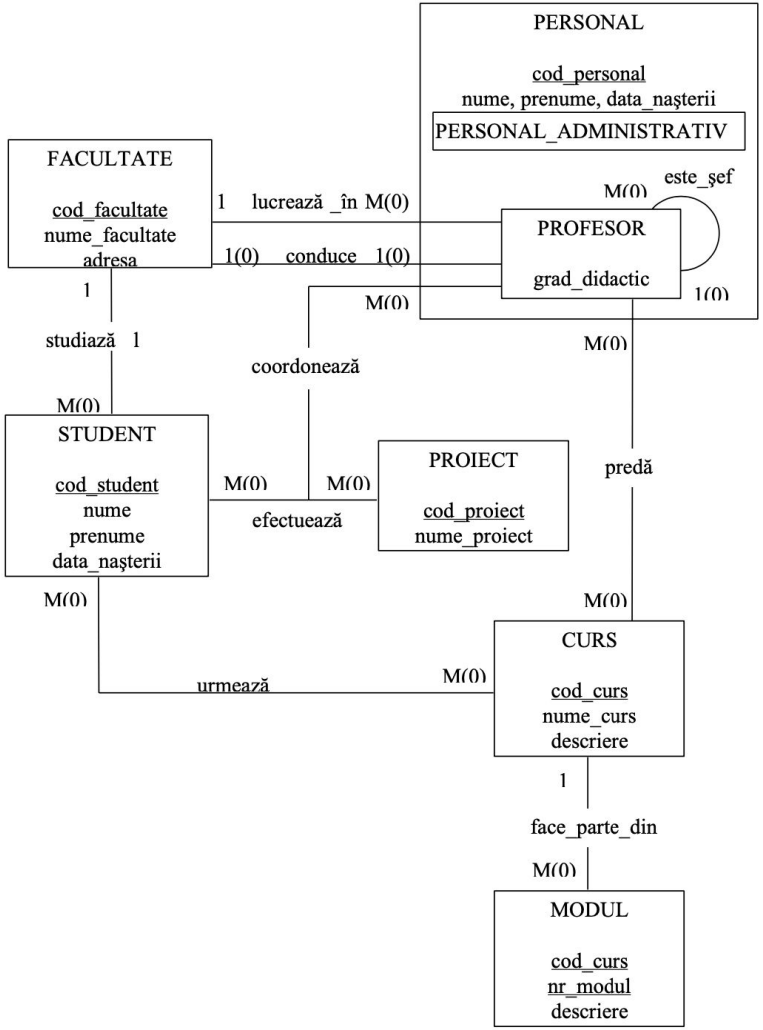
1. **Crearea schemei conceptuale.** Este un design de nivel înalt care descrie datele şi relaţiile necesare pentru execuţia operaţiilor necesare, fiind independent de orice model de baze de date.
2. **Crearea design-ului logic al bazei de date.** La sfârşitul acestei etape vom avea un număr de tabele care vor permite stocarea şi manipularea corectă a tuturor datelor necesare sistemului.
3. **Crearea design-ului fizic al bazei de date.** În această etapă designul logic este transformat într-o structură fizică eficientă.

1. Schema conceptuală

- **Etapele obţinerii modelului entitate-legătură**:
  - Identificarea entităţilor sistemului.
  - Identificarea relaţiilor sistemului şi stabilirea cardinalităţii acestora.
  - Identificarea atributelor entităţilor şi relaţiilor sistemului.
  - Stabilirea cheilor primare ale entităţilor.
  - Trasarea diagramei entitate-legătură.

Exemplu de ERD

Diagrama entitate-legătură a sistemului prezentat ca exemplu în cursul 3.

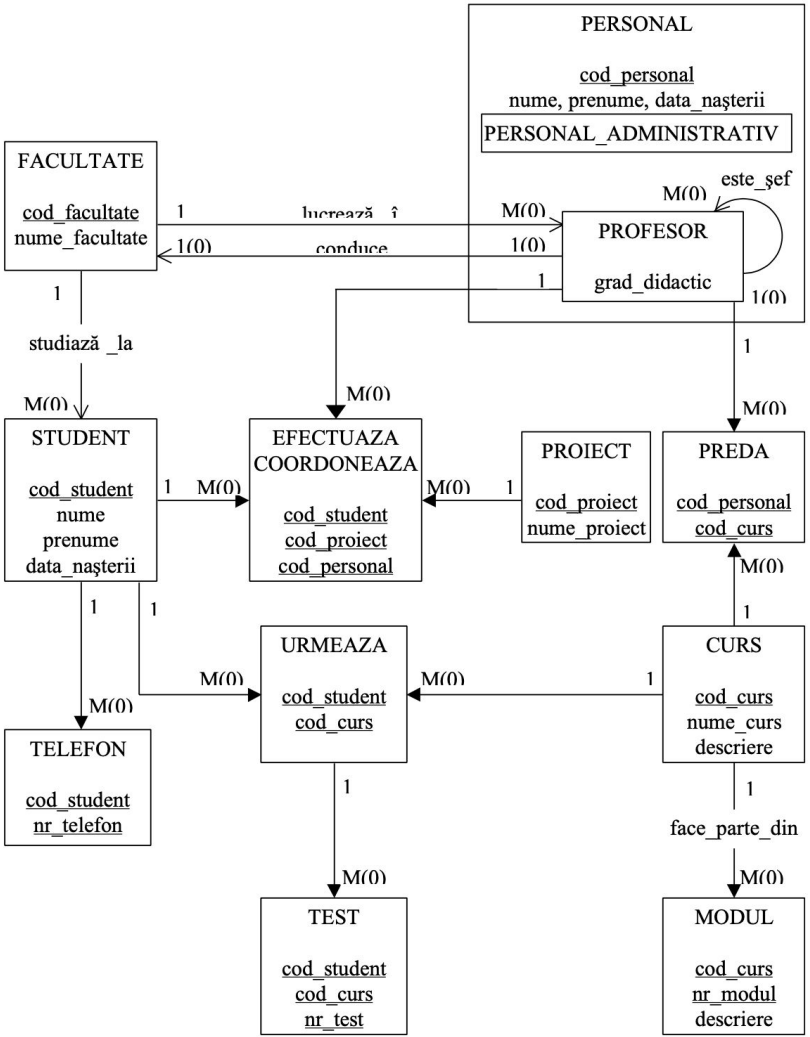


2. Design logic

- Se porneşte de la schema conceptuală, mai precis de la modelul entitate-legătură, şi se încearcă reprezentarea entităţilor şi a legăturilor sub formă de tabele relaţionale.

Exemplu de diagramă logică

Diagrama logică a BD pentru sistemul descris ca exemplu în cursul 3.



Exemplu de diagramă logică - cont.

➢ **Tabelele asociate acestei diagrame sunt următoarele:**

PERSONAL (cod_personal, nume, prenume, data_nastere, sex, stare_civila)
PERSONAL_ADMINISTRATIV (*cod_personal*, profesie, funcţie)
PROFESOR (*cod_personal*, grad_didactic, titlu, *sef*, ore_predate, data_angajării, *cod_facultate*)
CURS (cod_curs, nume_curs, descriere, nr_ore)
PREDA (*cod_personal*, *cod_curs*)
MODUL (*cod_curs*, nr_modul, descriere)
FACULTATE (cod_facultate, nume_facultate, localitate, strada, nr, cod_postal *cod_decan*)

Exemplu de diagramă logică - cont.

STUDENT (cod_student, nume, prenume, data_nasterii, tara, localitate, strada, nr, cod_postal, studii_anterioare)
TELEFON (*cod_student*, nr_telefon, tip_telefon)
PROIECT (cod_proiect, nume_proiect, domeniu)
EFECTUEAZA_COORDONEAZA (*cod_student*, *cod_proiect*, *cod_personal*)
URMEAZA (*cod_student*, *cod_curs*, nota_examen, nota_restantă, observatii)
TEST (*cod_student*, *cod_curs*, nr_test, nota_test, observatii)

➢ **Notă:** Atributele subliniate constituie CP a tabelului iar cele italice constituie chei străine.

2. Design fizic

- aka Normalizarea bazei de date.

Normalizarea bazei de date

- În general, pt. proiectare se creează mai întâi schema conceptuală care este transformată într-un design logic.
- În trecut, în locul acestei tehnici se utiliza o altă tehnică numită normalizare.
- **Normalizarea** constă din descompunerea unui tabel relațional în mai multe tabele care satisfac anumite reguli și stochează aceleași date ca și cel inițial.
- ∃ 6 forme normale: prima formă normală, a doua formă normală, a treia formă normală, forma normală Boyce-Codd, a patra formă normală şi a cincea formă normală.

Dependențe (nu dependințe!) funcționale

➢ Fie R un tabel relațional și X şi Y 2 submulţimi de coloane ale lui R.
Vom spune că **X determină funcţional pe Y** sau **Y depinde funcţional de X** dacă ∄ 2 rânduri în tabelul R care să aibă aceleaşi valori pentru coloanele din X şi să aibă valori diferite pentru cel puţin o coloană din Y.
Cu alte cuvinte, o valoare a lui X determină în mod unic o valoare a lui Y, adică ∀ 2 rânduri din R care au aceeaşi valoare pentru X trebuie să ia aceeaşi valoare pentru Y.
Notaţie: **X** → **Y**. X se va numi **determinant**, iar Y **determinat**.
Spunem că dependenţa X → Y este **trivială** dacă toate elementele lui Y sunt şi elemente ale lui X, Y ⊆ X.

- În curs simbolul ∀ poate însemna oricare, orice sau pentru orice, după caz.

Exemplu

➢ VÂNZĂRI (cod_client, nume_client, nr_telefon, cod_comandă, data, cod_articol, nume_articol, cost_articol, cantitate)

VÂNZĂRI

cod_ client	nume_ client	nr_ telefon	cod_ comandă	data	cod_ articol	nume_ articol	cost_ articol	cantitate
A1	Popescu	3215576	C1	12.05.99	P1	cămaşă	100.000	2
A1	Popescu	3215576	C1	12.05.99	P3	tricou	50.000	1
A2	Ionescu	2325587	C2	13.05.99	P1	cămaşă	100.000	3
A2	Ionescu	2325587	C2	13.05.99	P3	tricou	50.000	2
A2	Ionescu	2325587	C2	13.05.99	P2	pantaloni	200.000	1
A1	Popescu	3215576	C3	14.05.99	P3	tricou	50.000	3
A3	Georgescu	4555895	C4	14.05.99	P1	cămaşă	100.000	1

Exemplu - cont.

- Dependențe funcționale:
  {cod_articol} → {nume_articol, cost_articol}
  {cod_comandă} → {data, cod_client, nume_client, nr_telefon}
  {cod_client} → {nume_client, nr_telefon}

- **Notă**: dependenţa {cod_comandă} → {nume_client, nr_telefon} poate fi dedusă din dependenţele {cod_comandă} → {cod_client} și {cod_client} → {nume_client, nr_telefon}. Astfel de dependenţe se numesc **dependenţe tranzitive** şi vor fi definite mai riguros ulterior.

Prima formă normală (1NF)

- ➢ Un **tabel relaţional este în prima formă normală (1NF)** dacă fiecărei coloane îi corespunde o valoare indivizibilă (atomică), deci ∀ valoare nu poate să fie o mulţime sau un tuplu de valori. În plus, nu pot să apară grupuri de atribute repetitive.

- ➢ Tabelul VÂNZĂRI se află în prima formă normală.

- NF reprezintă abrevierea din lb. eng. pentru normal form

Exemple

- Pentru o coloană care conţine date calendaristice, sub forma **zz-ll-aa** (doi digiţi pentru zi, doi pentru lună şi doi pentru an), se consideră că **valoarea** respectivă **nu se poate descompune** în ziua, luna şi anul corespunzătoare acestei valori.

- Dacă vom considera că **adresa** este un atribut compus format din componentele ţară, oraş, stradă, număr şi cod, fiecare având semnificaţie proprie şi putând fi folosite independent la interogarea bazei de date, atunci adresa va fi reprezentată în tabel prin 5 coloane în loc de una.

Exemple - cont.

- un tabel aflat în 1NF nu poate conţine atribute sau grupuri de atribute repetitive.
  De exemplu, în tabelul următor, dacă un student poate avea **mai multe numere de telefon**, atunci coloanele “telefon1”, “telefon2”, “telefon3” constituie un **grup de atribute repetitive**.
  Deci tabelul STUDENT nu este în prima formă normală, el conţinând atât un atribut compus (“adresa”) cât şi un grup de atribute repetitive.

Exemple - cont.

STUDENT

cod_ student	nume	prenume	adresa	telefon1	telefon2	telefon3	materia	nota
101	Ionescu	Vasile	Romania, Bucureşti, Str. Polizu 5, 7355	6245981	3215678	092659019	Drept	10
101	Ionescu	Vasile	Romania, Bucureşti, Str. Polizu 5, 7355	6245981	3215678	092659019	Engleza	8
102	Pop	Costică	Romania, Cluj, Str. Unirii 7, 3551	3215469			Fizică	8

➢ **Algoritmul 1NFA** permite aducerea unei relaţii în 1NF prin **eliminarea atributelor compuse şi a celor repetitive**.

Algoritmul 1NFA

1. Se înlocuiesc în tabel coloanele corespunzătoare atributelor compuse cu coloane ce conţin componentele elementare ale acestora.
2. Se plasează grupurile de atribute repetitive, fiecare în câte un nou tabel.
3. Se introduce în fiecare tabel nou creat la pasul 2 CP a tabelului din care a fost extras atributul respectiv. Prin urmare, în tabelul nou creat atributele introduse vor fi chei străine ce fac referinţă la tabelul din care au fost extrase.
4. Se stabileşte CP a fiecărui nou tabel creat la pasul 2. Aceasta va fi creată din chei străine introdusă la pasul 3 plus una sau mai multe coloane adiţionale.

Exemple - cont.

STUDENT

cod_ student	nume	prenume	adresa	telefon1	telefon2	telefon3	materia	nota
101	Ionescu	Vasile	Romania, Bucureşti, Str. Polizu 5, 7355	6245981	3215678	092659019	Drept	10
101	Ionescu	Vasile	Romania, Bucureşti, Str. Polizu 5, 7355	6245981	3215678	092659019	Engleza	8
102	Pop	Costică	Romania, Cluj, Str. Unirii 7, 3551	3215469			Fizică	8

TELEFON

cod_ student	telefon
101	6245981
101	3215678
101	092659019
102	3215469

Observații

- 1NF este o cerinţă minimală a tuturor sistemelor relaţionale.
  Sistemele de baze de date care nu respectă nici măcar această formă nu pot fi numite relaţionale.
- Tabelele aflate în 1NF permit o referire simplă a datelor prin indicarea numelui tabelului, a coloanei şi a cheii rândului din care face parte informaţia respectivă.
- Operatorii pentru aceste tabele sunt mai simpli şi permit definirea unor tehnici de proiectare şi utilizare a bazelor de date.

A doua formă normală (2NF)

- Fie R un tabel relaţional şi fie X şi Y două submulţimi de coloane ale lui R.
  O **dependenţă funcţională** $X \rightarrow Y$ se numeşte **totală** dacă
  $\forall$ subset de coloane Z al lui X, $Z \subseteq X$, dacă $Z \rightarrow Y$ atunci $Z = X$.
  Cu alte cuvinte, $\nexists$ nici un subset Z al lui X, $Z \neq X$, pentru care $Z \rightarrow Y$.
  O dependență funcţională care nu este totală se numeşte **parţială**.

A doua formă normală (2NF) - cont.

- Un **tabel relaţional R este în a doua formă normală (2NF)** dacă şi numai dacă:
  - R este în 1NF.
  - ∀ coloană care depinde parţial de o cheie a lui R este inclusă în acea cheie.
- Cu alte cuvinte, a doua formă normală **nu permite dependenţe funcţionale parţiale faţă de cheile tabelului**, cu excepţia dependenţelor triviale, de incluziune.
  Deci tabelul VÂNZĂRI nu este în 2NF, coloanele “nume_articol” şi “cost_articol” depinzând parţial de CP a tabelului.

Regulă de descompunere

- Pentru a obţine tabele relaţionale în 2NF, tabelul iniţial se descompune fără pierdere de informaţie astfel:
  Fie **R(K1, K2, X, Y)** un tabel relaţional unde K1, K2, X şi Y sunt submulţimi de coloane ale lui R a.î. K1 ∪ K2 este o cheie a lui R, iar **K1 → X** este o dependenţă funcţională totală. Dacă X ⊂ K1 atunci tabelul este deja în 2NF, altfel tabelul R poate fi descompus prin proiecţie în **R1(K1, K2, Y)** - având cheia K1 ∪ K2 - şi **R2(K1, X)** - având cheia K1.

Regulă de descompunere - cont.

- Această descompunere conservă nu numai datele, ci şi dependenţele funcţionale, atât determinantul cât şi determinatul dependenţei eliminate regăsindu-se în tabelul nou creat.
- Folosind această regulă, **algoritmul 2NFA permite** aducerea în 2NF a unui tabel relaţional aflat în 1NF prin **eliminarea dependenţelor funcţionale parţiale**.

Algoritmul 2NFA

1. $\forall$ coloană X care depinde funcţional parţial de o cheie K, **K $\rightarrow$ X**, şi care nu este inclusă în K, se determină K1 $\subset$ K un subset al lui K, astfel încât **dependenţa K1 $\rightarrow$ X** este **totală** şi se creează un nou tabel **R1(K1, X)**, adică un tabel format din determinantul (K1) şi determinatul (X) acestei relaţii.

2. Dacă în tabelul R $\exists$ mai multe dependenţe totale ca mai sus cu acelaşi determinant, atunci pentru acestea se creează un singur tabel format din determinant - luat o singură dată - şi din determinaţii dependenţelor considerate.

Algoritmul 2NFA - cont.

3. Se elimină din tabelul iniţial R toate coloanele, X, care formează determinatul dependenţei considerate.

4. Se determină CP a fiecărui tabel nou creat, R1. Aceasta va fi K1, determinantul dependenţei considerate.

5. Dacă noile tabele create conţin alte dependenţe parţiale, atunci se merge la pasul 1, altfel algoritmul se termină.

Exemplu

- Pentru tabelul VÂNZĂRI CP este {cod_comandă, cod_articol}.
  Atributul “nume_articol” şi “cost_articol” depind funcţional parţial de această CP şi depind funcţional total de atributul “cod_articol” conţinut în CP.
  Acelaşi lucru se întâmplă şi cu atributele “data”, “cod_client”, “nume_client” şi “nr_telefon” care depind funcţional total numai de atributul “cod_comandă” conţinut în CP.

  Prin urmare vom avea următoarele **dependenţe totale**:
  {cod_articol} → {nume_articol, cost_articol}
  {cod_comandă} → {data, cod_client, nume_client, nr_telefon}

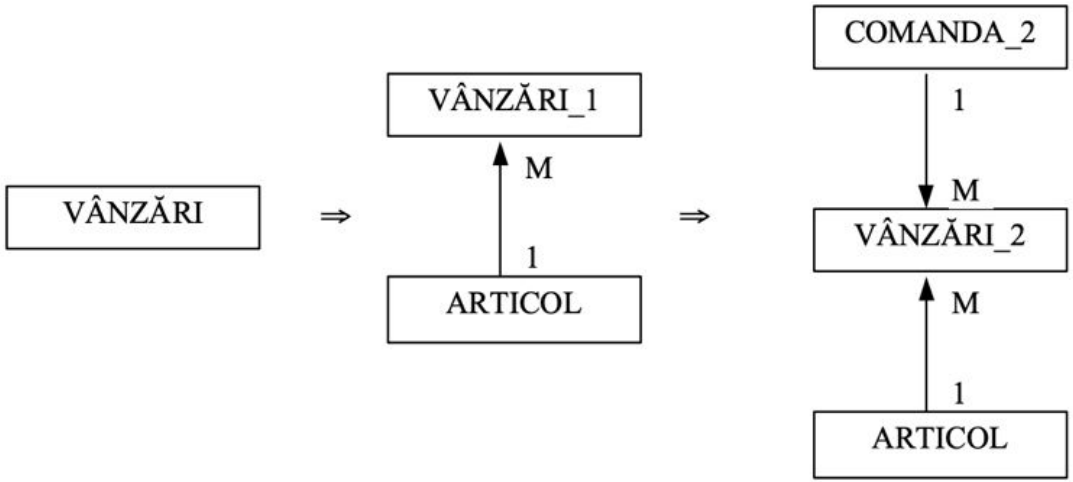
Exemplu - cont.

- Tabelul VÂNZĂRI (cod_client, nume_client, nr_telefon, cod_comandă, data, cod_articol, nume_articol, cost_articol, cantitate) va fi descompus în

  VÂNZĂRI_1 (cod_client, nume_client, nr_telefon, cod_comandă, data, cod_articol, cantitate)
  şi
  ARTICOL (cod_articol, nume_articol, cost_articol).

  Tabelul VÂNZĂRI_1 poate fi descompus în
  VÂNZĂRI_2 (cod_comandă, cod_articol, cantitate) şi
  COMANDA_2 (cod_comandă, data, cod_client, nume_client, nr_telefon).

Exemplu - cont.



- Un tabel care are CP formată dintr-un singur atribut este automat în 2NF.
  Prin urmare, algoritmul 2NFA nu se poate aplica decât în cazul în care CP a unui tabel este o cheie compusă.

A treia formă normală (3NF)

- Deşi tabelul COMANDA_2 este în 2NF, se observă că încă mai există redundanţă în date - tuplul (A1, Popescu, 3215576) apare de două ori.
  Intuitiv, aceasta se explică prin faptul că atributele “nume_client” şi “nr_telefon” depind indirect de CP a tabelului, dependenţa făcându-se prin intermediul atributului “cod_client”.
  Aceste dependenţe indirecte vor fi îndepărtate în 3NF.

A treia formă normală (3NF) - cont.

- Un **tabel relaţional R este în a treia formă normală (3NF)** dacă şi numai dacă:
  - R este în 2NF.
  - ∀ coloană A neconţinută în nici o cheie a lui R, dacă ∃ un set de coloane X a.î. X → A, atunci fie X conţine o cheie a lui R, fie A este inclusă în X.
- A doua condiţie din definiţie **interzice dependenţele funcţionale totale faţă de alte coloane în afara celor care constituie chei ale tabelului**.
  Prin urmare, un tabel este în 3NF dacă ∀ **coloană care nu este conţinută într-o cheie depinde de cheie, de întreaga cheie şi numai de cheie**.

A treia formă normală (3NF) - cont.

- În mod evident tabelul COMANDA_2 nu este în 3NF existând dependenţele
  {cod_client} → {nume_client} şi
  {cod_client} → {nr_telefon}.

  Pe de altă parte, tabelele VÂNZĂRI_2 şi ARTICOL sunt în 3NF.

A treia formă normală (3NF) - cont.

- Cea de-a doua condiţie din definiţia de mai înainte se formulează folosind noţiunea de **dependenţă tranzitivă**.
  Fie R un tabel relaţional, X o submulţime de coloane a lui R şi A o coloană a lui R.
  Spunem că **A este dependentă tranzitiv de X** dacă ∃ o submulţime de coloane Y care nu include A şi nu determină funcţional pe X astfel încât **X** → **Y** şi **Y** → **A**.
  Dacă în această definiţie se doreşte să se evidenţieze şi Y atunci se spune că A depinde funcţional de X prin intermediul lui Y şi se scrie X → Y → A.

A treia formă normală (3NF) - cont.

- De exemplu, în tabelul COMANDA_2 coloanele “nume_client” şi “nr_telefon” depind tranzitiv de CP “cod_comandă” prin intermediul coloanei “cod_client”.
  Folosind această definiţie, condiţia ca un tabel să fie în 3NF se poate reformula astfel:

- Un **tabel relaţional R este în a treia formă normală (3NF)** dacă şi numai dacă:
  - R este în 2NF.
  - ∀ coloană neconţinută în nici o cheie a lui R nu este dependentă tranzitiv de nici o cheie a lui R.

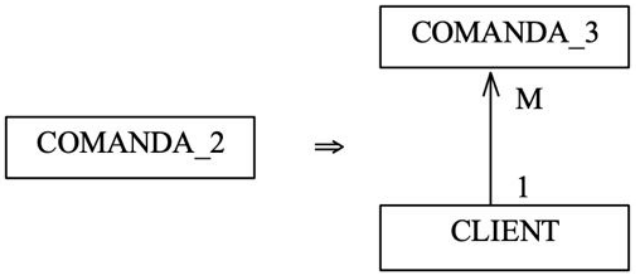
Regulă de descompunere

- Pentru a obţine tabele relaţionale în 3NF, tabelul iniţial se descompune fără pierdere de informaţie după următoarele reguli.
  Fie **R(K, X, Y, Z)** un tabel relaţional unde K este o cheie a lui R, iar X, Y şi Z sunt submulţimi de coloane ale lui R.
  - Dacă există **dependenţa tranzitivă K → X → Y**, atunci R se poate descompune în **R1(K, X, Z)** - având cheia K - şi **R2(X, Y)** - având cheia X.
  - Dependenţa tranzitivă poate fi mai complexă. Fie **K1 ⊂ K** o parte a cheii K astfel încât există dependenţa tranzitivă **K → K1 ∪ X → Y**. În acest caz, R poate fi descompus în **R1(K, X, Z)** - având cheia K - şi **R2(K1, X, Y)** - având cheia K1 ∪ X.

Regulă de descompunere - cont.

- Descompunerile corespunzătoare regulilor de mai înainte conservă nu numai datele, ci şi dependenţele funcţionale, determinantul şi determinatul dependenţelor eliminate regăsindu-se în tabelele nou create.

- Un exemplu de aplicare a primei reguli este descompunerea tabelului COMANDA_2 în COMANDA_3 (cod_comandă, data, cod_client) şi CLIENT (cod_client, nume_client, nr_telefon)

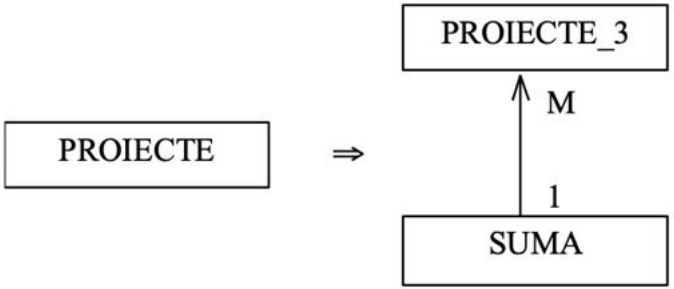


Exemplu

- Pentru a exemplifica cea de-a doua regulă, considerăm
  PROIECTE (cod_angajat, cod_proiect, rol_în_proiect, suma_obţinută),
  care stochează date privind repartizarea pe proiecte a angajaţilor unei firme.
  Presupunem că suma obţinută de un angajat depinde de proiectul respectiv şi de rolul angajatului în acel proiect, deci avem dependenţa
  {cod_proiect, rol_în_proiect} → {suma_obţinută}.

Exemplu - cont.

- Aplicând regula a doua, tabelul PROIECTE se descompune în:
  PROIECTE_3 (cod_angajat, cod_proiect, rol_în_proiect) şi
  SUMA (cod_proiect, rol_în_proiect, suma_obţinută).



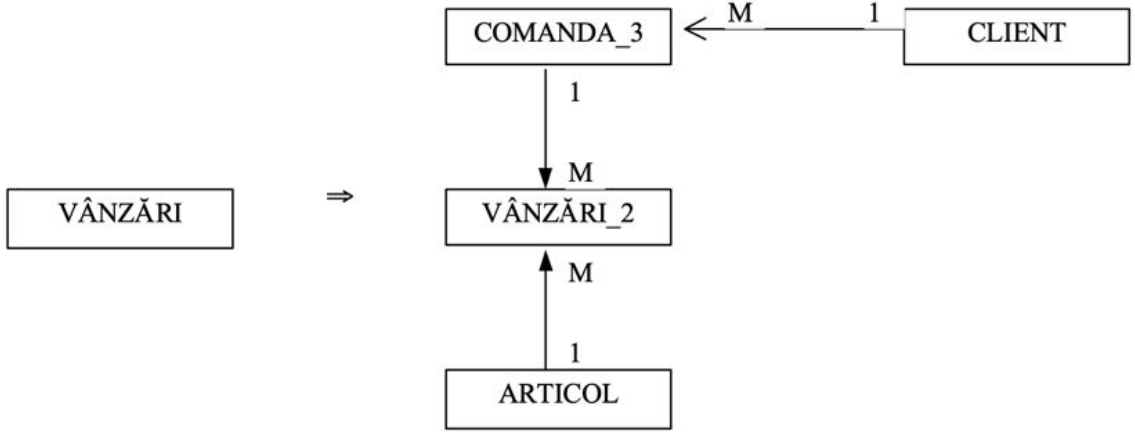
- Folosind această regulă, algoritmul 3NFA permite aducerea în 3NF a unui tabel relaţional aflat în 2NF prin **eliminarea dependenţelor funcţionale tranzitive**.

Algoritmul 3NFA

1. Pentru fiecare dependenţă funcţională tranzitivă **K** → **X** → **Y** se transferă coloanele din X şi Y într-o nouă relaţie.
2. Se determină CP a fiecărei noi relaţii create la pasul 1, aceasta fiind formată din coloanele din X.
3. Se elimină din relaţia principală coloanele din Y.
4. Dacă tabelele rezultate conţin alte dependenţe tranzitive, atunci se merge la pasul 1, altfel algoritmul se termină.

Exemplu

- Aplicând algoritmii 2NFA şi 3NFA, tabelul VÂNZĂRI a fost descompus în tabelele VÂNZĂRI_2, ARTICOL, COMANDA_3 şi CLIENT, care sunt toate în 3NF.
  În aceste tabele nu mai există nici redundanţă în date şi nici anomalii de actualizare.



Forma normală Boyce-Codd (BCNF)

- Nu toate tabelele aflate în 3NF sunt lipsite de redundanţă în date şi anomalii de actualizare. Pentru a ilustra această situaţie să considerăm următorul exemplu.

  O companie de transporturi efectuează curse, în care poate folosi unul sau mai mulţi şoferi - de exemplu prima jumătate a cursei conduce un şofer, cea de-a doua alt şofer - şi mai multe dintre autobuzele aflate în dotare, cu condiţia ca într-o cursă un şofer să conducă un singur autobuz.
  Pe de altă parte însă, un autobuz este repartizat unui şofer şi deci nu poate fi condus decât de acesta.

Forma normală Boyce-Codd (BCNF) - cont.

- Această situaţie poate fi modelată printr-un tabel TRANSPORTURI (cod_cursă, cod_şofer, cod_autobuz, loc_plecare, loc_sosire).
- În acest tabel avem dependenţele:
  {cod_cursă, cod_şofer} → {cod_autobuz, loc_plecare, loc_sosire}
  {cod_autobuz} → {cod_şofer},
  iar cheile tabelului sunt {cod_cursă, cod_şofer} şi {cod_cursă, cod_autobuz}.
- Tabelul TRANSPORTURI este în 3NF şi totuşi în acest tabel există redundanţă în date datorită dependenţei {cod_autobuz} → {cod_şofer}.

Forma normală Boyce-Codd (BCNF) - cont.

- Forma Boyce-Codd elimină acest tip de redundanţe.
  Intuitiv, un tabel R este în BCNF dacă fiecare determinant al unei dependenţe funcţionale este cheie candidată a lui R.
  Tabelul TRANSPORTURI de mai înainte nu este în BCNF, “cod_autobuz” nefiind o cheie a lui R.
- Un **tabel relaţional este în forma normală Boyce-Codd (BCNF)** $\Leftrightarrow \forall$ dependenţă funcţională totală $X \rightarrow A$, unde X este un subset de coloane iar A o coloană neconţinută în X, X este o cheie a lui R.

Forma normală Boyce-Codd (BCNF) - cont.

- ∀ tabel care are cel mult 2 coloane este în BCNF.
  ∀ tabel relaţional se poate descompune fără pierdere de informaţie în tabele aflate în BCNF, dar nu acelaşi lucru se poate spune despre descompunerea cu păstrarea dependenţelor funcţionale, după cum vom vedea în continuare.

Algoritmul BCNFA

1. ∀ dependenţă **non-cheie X → Y**, unde X şi Y sunt subseturi de coloane ale lui R, se creează 2 tabele. Una dintre ele va fi formată din coloanele {X, Y}, iar cealaltă va fi formată din toate coloanele iniţiale, mai puţin coloanele Y.
2. Dacă tabelele rezultate conţin alte dependenţe non-cheie, atunci se merge la pasul 1, altfel algoritmul se termină.

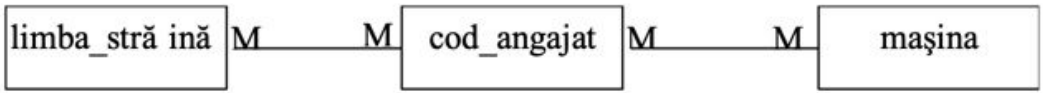
Exemplu

- Aplicând acest algoritm, tabelul TRANSPORTURI se va descompune în tabelele TRANSPORTURI_BC (cod_cursă, cod_autobuz, loc_plecare, loc_sosire) şi AUTOBUZ (cod_autobuz, cod_şofer).

- Descompunerea s-a făcut fără pierdere de informaţie, dar a fost pierdută dependenţa funcţională {cod_cursă, cod_şofer} → {cod_autobuz}.

A patra formă normală (4NF)

- Dacă BCNF elimină redundanţele datorate dependenţelor funcţionale, **4NF determină redundanţele datorate dependenţelor multivaloare**.
- Pentru a ilustra acest tip de redundanţe, să considerăm tabelul ANGAJAŢI (cod_angajat, limba_străină, maşina) aflat în BCNF.
- Un angajat poate cunoaşte mai multe limbi străine şi poate avea mai multe maşini, dar ∄ nici o legătură între limba străină şi maşină. Cu alte cuvinte, redundanţa datelor din tabelul ANGAJAŢI este cauzată de existenţa a două relaţii N:M independente. 4NF va înlătura aceste relaţii N:M independente.



A patra formă normală (4NF) - cont.

- Fie R un tabel relaţional, X şi Y două submulţimi de coloane ale lui R şi **Z = R - X - Y** mulţimea coloanelor din R care nu sunt nici în X nici în Y.
  Spunem că **∃ o dependenţă multivaloare Y de X** sau că **X determină multivaloare pe Y**, şi notăm **X →→ Y**, dacă, ∀ valoare a coloanelor lui X, sunt asociate valori pentru coloanele din Y care nu sunt corelate în nici un fel cu valorile coloanelor lui Z.

A patra formă normală (4NF) - cont.

- Cu alte cuvinte $X \twoheadrightarrow Y \Leftrightarrow \forall$ u şi v, 2 rânduri ale lui R cu u(X) = v(X), $\exists$ s şi t, 2 rânduri ale lui R a.î.
  s(X) = u(X), s(Y) = u(Y), s(Z) = v(Z) şi a.î.
  t(X) = u(X), t(Y) = v(Y), t(Z) = u(Z), unde prin u(X) am notat valoarea coloanelor X corespunzătoare rândului u, etc.
  În mod evident, dacă $X \twoheadrightarrow Y$ atunci şi $X \twoheadrightarrow Z$.
- Dependenţa multivaloare se mai numeşte şi **multidependenţă**.

	X	Y	Z
u	x	y1	z1
v	x	y2	z2
s	x	y1	z2
t	x	y2	z1

A patra formă normală (4NF) - cont.

- În tabelul ANGAJAŢI avem dependenţele multivaloare
  cod_angajat →→ limba_străină şi
  cod_angajat →→ maşina.
  Pe de altă parte însă, nici “limba_străină” şi nici “maşina” nu depinde funcţional de “cod_angajat”.

A patra formă normală (4NF) - cont.

- Un **tabel relaţional R este în a patra formă normală (4NF)** dacă şi numai dacă:
  - R este în BCNF.
  - $\forall$ dependenţă multivaloare $X \twoheadrightarrow Y$ este de fapt o dependenţă funcţională $X \rightarrow Y$.
- Condiţia a doua arată că dacă $\exists$ o dependenţă multivaloare $X \twoheadrightarrow Y$, atunci $\forall$ coloană, A a lui R va depinde funcţional de coloanele din X, $X \rightarrow A$ - aceasta deoarece
  existenţa unei dependenţe multivaloare $X \twoheadrightarrow Y$ implică şi
  existenţa unei dependenţe multivaloare $X \twoheadrightarrow (R - X - Y)$.
  Ţinând cont de faptul că R este în BCNF, înseamnă că $\exists$ o cheie candidată a lui R inclusă în X.
- Un **tabel relaţional R este în a patra formă normală (4NF)** $\Leftrightarrow$
  $\forall$ dependenţă multivaloare $X \twoheadrightarrow Y$ $\exists$ o cheie a lui R inclusă în X.

Regulă de descompunere

- Un tabel poate fi adus în 4NF prin descompunere fără pierdere de informaţie astfel:
  Fie **R(X, Y, Z)** un tabel relaţional în care ∃ o **dependenţă multivaloare X →→ Y** astfel încât X nu conţine nici o cheie a lui R.
  Atunci tabelul R poate fi descompus prin proiecţie în două tabele **R1(X, Y)** şi **R2(X, Z)**.

- Aplicând această regulă tabelul ANGAJAŢI se descompune în tabelele
  ANGAJAŢI_4A (cod_angajat, limba_străină) și
  ANGAJAŢI_4B (cod_angajat, maşina).

Algoritmul 4NFA

1. Se identifică **dependenţele multivaloare X →→ Y** pentru care X şi Y nu conţin toate coloanele lui R şi X nu conţine nici o cheie a lui R.
   Se poate presupune că X şi Y sunt disjuncte datorită faptului că din X →→ Y rezultă X →→ (Y - X).
2. Se înlocuieşte tabelul iniţial R cu două tabele, primul format din coloanele {X, Y}, iar celălalt din toate coloanele iniţiale, mai puţin coloanele Y.
3. Dacă tabelele rezultate conţin alte dependenţe multivaloare, atunci se face transfer la pasul 1, altfel algoritmul se termină.

A cincea formă normală (5NF)

- A cincea formă se întâlneşte destul de rar în practică, ea având mai mult valoare teoretică. Dacă în a patra formă normală sunt eliminate relaţiile N:M independente, a cincea formă normală are ca scop **eliminarea relaţiilor N:M dependente**.

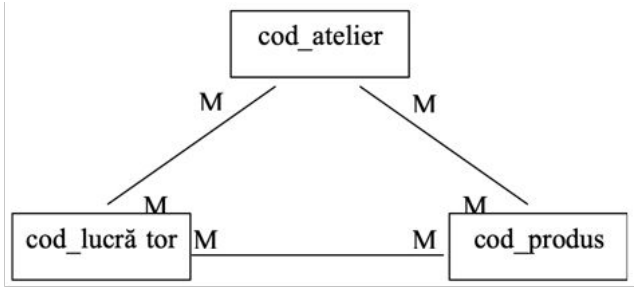
  Redundanţele datorate unor astfel de relaţii pot fi înlăturate prin descompunerea tabelului în 3 sau mai multe tabele.

Exemplu

- Considerăm tabelul
  LUCRĂTOR_ATELIER_PRODUS (cod_lucrător, cod_atelier, cod_produs), aflat în 4NF.
  Aparent acest tabel este ilustrarea unei relaţii de tip 3 care există între
  lucrător, atelier şi produs.
  Dacă însă presupunem că între
  lucrător şi atelier, lucrător şi produs, atelier şi produs există relaţii N:M,
  atunci în tabel pot exista redundanţe în date care pot fi înlăturate prin descompunerea tabelului LUCRĂTORI în 3 tabele:
  LUCRĂTOR_ATELIER (cod_lucrător, cod_atelier),
  LUCRĂTOR_PRODUS (cod_lucrător, cod_produs) şi
  ATELIER_PRODUS (cod_atelier, cod_produs).

LUCRĂTOR_ATELIER_PRODUS

Cod_lucrător	cod_atelier	cod_produs
L1	A1	P1
L2	A1	P2
L2	A1	P1
L2	A2	P1



LUCRĂTOR_ATELIER

Cod_lucrător	cod_atelier
L1	A1
L2	A1
L2	A2

LUCRĂTOR_PRODUS

Cod_lucrător	cod_produs
L1	P1
L2	P2
L2	P1

ATELIER_PRODUS

Cod_atelier	cod_atelier
A1	P1
A1	P2
A2	P1

R_12

Cod_lucrător	cod_atelier	cod_produs
L1	A1	P1
L2	A1	P2
L2	A1	P1
L2	A2	P2
L2	A2	P1

R_13

Cod_lucrător	cod_atelier	cod_produs
L1	A1	P1
L1	A1	P2
L2	A1	P2
L2	A1	P1
L2	A2	P1

R_23

Cod_lucrător	cod_atelier	cod_produs
L1	A1	P1
L1	A2	P1
L2	A1	P2
L2	A1	P1
L2	A2	P1

Exemplu - cont.

- Tabelul LUCRĂTOR_ATELIER elimină redundanţa (L2, A1), tabelul LUCRĂTOR_PRODUS elimină redundanţa (L2, P1), în timp ce tabelul ATELIER_PRODUS elimină redundanţa (A1, P1).
- Tabelul iniţial LUCRĂTOR_ATELIER_PRODUS nu poate fi reconstituit din compunerea a doar două din tabelele componente, unde tabelul **R_12** rezultă din compunerea LUCRĂTOR_ATELIER şi LUCRĂTOR_PRODUS, **R_13** rezultă din compunerea LUCRĂTOR_ATELIER şi ATELIER_PRODUS, iar **R_23** rezultă din compunerea LUCRĂTOR_PRODUS şi ATELIER_PRODUS.
- Tabelul iniţial LUCRĂTOR_ATELIER_PRODUS poate fi obţinut prin compunerea tuturor celor trei tabele componente, de exemplu el rezultă prin compunerea lui R_12 cu ATELIER_PRODUS.

Join-dependență

- Dependenţa funcţională şi dependenţa multivaloare, şi implicit regulile de descompunere pentru formele normale 1NF-4NF, permit **descompunerea prin proiecţie a unui tabel relaţional în 2 tabele relaţionale**.
  Totuşi, regulile de descompunere asociate acestor forme normale nu dau toate descompunerile posibile prin proiecţie a unui tabel relaţional.
  ∃ tabele care nu pot fi descompuse în 2 tabele, dar pot fi descompuse în 3 sau mai multe tabele fără pierdere de informaţie.
  Astfel de descompuneri, în 3 sau mai multe tabele, sunt tratate de 5NF.
  Pentru a arăta că un tabel se poate descompune fără pierderi de informaţie a fost introdus conceptul de **join-dependenţă** sau **dependenţă la compunere**.

Join-dependență - cont.

- Fie R un tabel relaţional şi R1, R2, ..., Rn o mulţime de tabele relaţionale care nu sunt disjuncte - au coloane comune - a.î. ∪ coloanelor din R1, R2, ..., Rn este mulţimea coloanelor din R.
  Se spune că R satisface **join-dependenţa *{R1, R2, ..., Rn}** dacă R se descompune prin proiecţie pe R1, R2, ..., Rn fără pierdere de informaţie, adică tabelul iniţial poate fi reconstruit prin compunerea naturală pe atribute comune ale tabelelor rezultate.

- În exemplul anterior, tabelul LUCRĂTOR_ATELIER_PRODUS satisface dependenţa de uniune *{LUCRĂTOR_ATELIER, LUCRĂTOR_PRODUS, ATELIER_PRODUS}.

Join-dependență - cont.

- Join-dependenţa este o generalizare a dependenţei multivaloare.
  Mai precis, **dependenţa multivaloare corespunde join-dependenţei cu două elemente**.
  Într-adevăr, dacă în relaţia **R(X, Y, Z)** avem multidependenţă **X →→ Y**, atunci avem şi join-dependenţă ***{(X ∪ Y), (X ∪ Z)}**.
  Invers, dacă avem join-dependenţă ***{R1, R2}**, atunci avem şi dependenţă multivaloare **(R1 ∩ R2) → (R1 - (R1 ∩ R2))**.

A cincea formă normală (5NF) - cont.

- Un **tabel relaţional R este în a cincea formă normală (5NF)** dacă şi numai dacă ∀ join-dependenţă *{R1, R2, ..., Rn} este consecinţa cheilor candidate ale lui R, adică fiecare dintre R1, R2, ..., Rn include o cheie candidată a lui R.

A cincea formă normală (5NF) - cont.

- ∀ tabel relaţional care este în 5NF este în 4NF deoarece ∀ dependenţă multivaloare poate fi privită ca un caz particular de dependenţă la uniune.
  Trecerea de la 4NF la 5NF constă în identificarea join-dependenţelor cu mai mult de 3 elemente şi descompunerea tabelului iniţial prin proiecţie pe aceste componente.
  5NF are o importanţă practică redusă, cazurile când apare în practică fiind extrem de rare.

- ∀ tabel relaţional poate fi descompus fără pierderi de informaţie într-o mulţime de tabele relaţionale care sunt în 5NF.
  Un tabel în 5NF nu conţine anomalii ce pot fi eliminate luând proiecţiile pe diferite submulţimi de coloane ale acestuia.

Concluzii

- Normalizarea este procesul de transformare a structurilor de date şi are ca scop **eliminarea redundanţelor** şi **promovarea integrităţii datelor**.
  Normalizarea este un pilon de bază al BD relaţionale.
  În general, un set de structuri de date nu sunt considerate relaţionale decât dacă sunt complet normalizate.
- **Normalizarea** datelor este împărţită în **6** etape, numite **forme normale (NF)**.
  Fiecare NF are asociat atât un criteriu cât şi un proces.
  Criteriul unei anumite NF este mai restrictiv decât al uneia inferioare, a. î.
  ∀ tabel relaţional care este într-o anumită NF este şi în NF inferioară.
  Procesul asociat unei NF se referă la trecerea unor structuri de date din NF inferioară în forma normală curentă.

Concluzii - cont.

- 1NF → 2NF elimină dependenţele funcţionale parţiale faţă de chei.
- 2NF → 3NF elimină dependenţele funcţionale tranzitive faţă de chei.
- 3NF → BCNF elimină dependenţele funcţionale pentru care determinantul nu este cheie.
- BCNF → 4NF elimină toate dependenţele multivaloare care nu sunt şi dependenţe funcţionale.
- 4NF → 5NF elimină toate join-dependenţele care nu sunt implicate de o cheie.

Concluzii - cont.

- În practică, cel mai adesea apar primele forme normale (1NF - 3NF), a. î. un tabel relaţional aflat în 3NF este de obicei complet normalizat.

  În special 5NF apare extrem de rar, având valoare practică foarte scăzută.

Denormalizare

- În principiu, denormalizarea este exact procesul invers normalizării.
- Este procesul de creştere a redundanţei datelor, care are ca scop creşterea performanţei sau simplificarea programelor de manipulare a datelor.
  Totuşi, trebuie reţinut că o denormalizare corectă **nu înseamnă nicidecum a nu normaliza structurile de date iniţiale**.
  Din contră, denormalizarea **are loc după ce structurile bazei de date au fost complet normalizate**, şi se face prin **selectarea strategică a acelor structuri unde denormalizarea aduce avantaje semnificative**.
  Pe de altă parte, orice denormalizare trebuie însoţită de introducerea de măsuri suplimentare, care să asigure integritatea datelor chiar şi în cazul unei BD care nu este complet nenormalizată.

Creșterea performanței

- Principalul obiectiv al denormalizării este creşterea performanţei programelor de manipulare a datelor.
  Una dintre cele mai întâlnite situaţii de acest gen este denormalizarea folosită pentru **operaţii sau calcule efectuate frecvent**.
- Să presupunem că pentru înregistrarea tranzacţiilor unui magazin care vinde articole la comandă se folosesc tabelele
  VÂNZĂRI_2 (cod_comandă, cod_articol, cantitate),
  ARTICOL (cod_articol, nume_articol, cost_articol),
  COMANDA_3 (cod_comandă, data, cod_client),
  CLIENT (cod_client, nume_client, nr_telefon)

Creșterea performanței - cont.

- Aceste tabele sunt în 5NF şi conţin toate datele necesare pentru obţinerea oricărui raport privind vânzările din magazin - de exemplu, în funcţie de articol, dată, cantitate, client, etc.
  Totuşi, să presupunem că majoritatea rapoartelor cerute de conducerea magazinului utilizează **cantitatea totală vândută într-o lună pentru fiecare articol**.
  Tabelele de mai înainte conţin toate datele necesare pentru obţinerea acestei informaţii, de exemplu ea se poate obţine printr-o simplă interogare SQL.
  Pe de altă parte însă, aceasta ar însemna ca **totalul pentru fiecare articol să fie recalculat de fiecare dată când este cerut**.

Creșterea performanței - cont.

- Deoarece este puţin probabil ca acest total să se schimbe după încheierea lunii respective, repetarea acestora ar putea fi înlocuită cu un tabel suplimentar
  ARTICOL_LUNA (cod_articol, luna, cantitate_totală)
  Datele conţinute în acest nou tabel sunt **redundante**, ele putând fi obţinute în orice moment din datele conţinute în tabelele VÂNZĂRI_2 şi COMANDA_3, dar prezintă avantajul că **folosirea** lor este **mai facilă şi mai rapidă decât repetarea calculelor**.
- Pe de altă parte însă, folosirea acestui tabel suplimentar are şi **dezavantaje**, putând duce la **pierderea integrităţii datelor**.
  Modificarea datelor din tabelele VÂNZĂRI_2 şi COMANDA_3 nu se reflectă automat în datele din noul tabel.

Creșterea performanței - cont.

- De exemplu, dacă după efectuarea calculelor şi inserarea totalului în noul tabel se efectuează o vânzare suplimentară, această vânzare nu este reflectată în datele din noul tabel.

  De aceea, în situaţiile în care BD nu este total normalizată, trebuie luate măsuri suplimentare pentru păstrarea integrităţii datelor.

Creșterea performanței - cont.

- De exemplu, pentru ca orice modificare a tabelelor iniţiale să fie reflectată automat în noul tabel, în Oracle se pot crea aşa numitele **triggere** (declanşatoare) ale bazei de date bazate pe tabelele VÂNZĂRI_2 şi COMANDA_3 care se declanşează după fiecare actualizare (INSERT, UPDATE, DELETE) a acestor tabele şi care actualizează datele din tabelul ARTICOL_LUNA. Aceste triggere pe bază, la rândul lor, vor încetini operaţiile de actualizare din tabelele pe care sunt bazate.

Creșterea performanței - cont.

- Datorită problemelor legate de redundanţă şi integritate, orice denormalizare trebuie să fie gândită cu atenţie pentru a fi siguri că avantajele acesteia în privinţa performanţelor şi a simplităţii programelor depăşesc efortul necesar pentru impunerea integrităţii.

- **În exemplul anterior, se pot crea tabele suplimentare pentru totaluri pe lună sau pe zi. Dacă aceste totaluri sunt utilizate rar, este mai convenabil ca ele să fie calculate direct din tabele normalizate, fără a mai introduce alte date redundante**.

- În concluzie, fiecare caz trebuie studiat cu atenţie pentru a vedea dacă problemele inerente asociate denormalizării (redundanţa, menţinerea integrităţii) sunt compensate de creşterea în performanţă pentru anumite situaţii.

Simplificarea codului

- Un alt motiv invocat pentru folosirea denormalizării este acela al simplificării codului pentru manipularea datelor.
  Cu alte cuvinte, pentru un dezvoltator, un singur tabel poate părea uneori mai uşor de utilizat decât două sau mai multe.
  De exemplu, să considerăm tabelul
  STOCURI(cod_depozit, cod_material, nume_material, cantitate)
  utilizat de o firmă pentru a înregistra cantităţile de materiale existente în fiecare din depozitele sale.
  Evident, acest tabel nu este în 2NF şi poate fi normalizat prin descompunerea în 2 tabele:
  STOCURI_2(cod_depozit, cod_material, cantitate)
  MATERIAL(cod_material, nume_material)

Simplificarea codului - cont.

- Totuşi, dacă se doreşte aflarea tuturor depozitelor în care există ciment, de exemplu, pentru dezvoltator este mai convenabil să folosească varianta nenormalizată, în acest caz interogarea SQL corespunzătoare fiind

```
SELECT cod_depozit, cantitate
FROM stocuri
WHERE nume_material = 'CIMENT';
```

- Evident, interogarea de mai sus este mai simplă decât varianta în care se folosesc cele două tabele normalizate

```
SELECT cod_depozit, cantitate
FROM stocuri_2, material
WHERE stocuri_2.cod_material = material.cod_material AND
      nume_material = 'CIMENT';
```

Simplificarea codului - cont.

- Pe de altă parte însă, diferenţa de performanţă (timp de execuţie) dintre cele două variante de interogări este neglijabilă, de multe ori chiar interogarea pe structuri normalizate va fi mai rapidă decât cealaltă, astfel încât singurul avantaj al folosirii structurii nenormalizate este simplitatea.
- Dacă pentru un dezvoltator simplificarea invocată mai sus este nesemnificativă - orice dezvoltator trebuie să fie capabil să scrie o interogare pe mai multe tabele de tipul celei de mai sus - problema se pune cu mai mare stringenţă în cazul în care utilizatorii pot să-şi scrie propriile interogări ale bazei de date.

Simplificarea codului - cont.

- În acest caz, pentru a păstra atât avantajele normalizării şi pentru a permite în acelaşi timp simplificarea interogărilor, se poate crea o vedere bazată pe cele două tabele normalizate

```
CREATE VIEW stocuri AS
SELECT cod_depozit, stocuri_2.cod_material, nume_material, cantitate
FROM stocuri_2, material
WHERE stocuri_2. cod_material = material.cod_material;
```

- O astfel de vedere “ascunde” tabelele normalizate şi permite utilizatorilor scrierea unor interogări mai simple.
  Pe de altă parte însă, performanţa programelor poate scădea mult în cazul vederilor complexe, în acest caz fiind uneori de preferat a se utiliza tabelele de bază pentru creşterea performanţei.

Concluzii

- Designul logic al BD are un **impact** profund asupra **performanţelor** şi **structurii sistemului**.
  Deşi este posibil a compensa un design prost prin scrierea unui volum mare de cod, acest lucru nu este în nici un caz de dorit, mai ales că în majoritatea cazurilor el va duce inevitabil la scăderea performanţelor sistemului.

- În mod ideal, dezvoltarea oricărei aplicaţii trebuie să înceapă de la o structură complet normalizată, care să constituie o temelie solidă a acesteia.

- Denormalizarea, dacă ea este necesară, trebuie făcută numai după ce se obţine o structură complet normalizată.
  Ca regulă generală, denormalizarea este folosită rar şi numai după o analiză atentă.

Temă

1. Se dă tabelul aflat în FN1:
   INFO_STUDENT(cod_student, nume_student, data_nastere, cod_facultate, nume_facultate, cod_curs, nume_curs, data_inceput, nr_examen, nota)
   Se presupune că:
   - ➢ Un student poate studia la o singură facultate;
   - ➢ Un student poate urma mai multe cursuri;
   - ➢ Data_inceput reprezintă data la care studentul a început cursul;
   - ➢ Între curs și facultate nu există legătură;
   - ➢ În cadrul unui curs, studentul dă mai multe examene (numerotate prin intermediul coloanei nr_examen). Nota reprezintă nota luată de student la examenul cu numărul nr_examen din cadrul cursului cu codul cod_curs.

Temă - cont.

Să se determine cheia primară a tabelului și să se aducă în 2NF și 3NF, specificându-se cheile primare ale tabelelor rezultate.
Notă: nu se punctează decât rezolvările în care se parcurg toți pașii asociați normalizării (i.e. identificarea dependențelor parțiale și transitive față de cheie, descompunerea în tabele pentru eliminarea acestor dependențe, etc.).

2. Să se ilustreze printr-un exemplu cum se transformă atributele repetitive (multivaloare) ale unei relații mulți-la-mulți (din modelul entitate-legătură) la crearea design-ului logic al unei baze de date relaționale.

3. Să se dea un exemplu de tabel relațional în care există o dependență multivaloare (multidependență) între atributele (coloanele) sale, care nu este dependență funcțională.

Bibliografie

F. Ipate, M. Popescu, Dezvoltarea aplicațiilor de baze de date în Oracle 8 și Oracle Forms 6, Editura ALL, 2000.