Indrumar Baze de Date – Capitolul 5

NORMALIZAREA RELATIILOR

Dependentele de date reprezinta constrângeri care se impun valorilor atributelor unei relatii si ele determina proprietatile relatiei în raport cu operatiile de inserare, stergere si actualizare a tuplurilor, proprietati care definesc gradul de normalizare (*forma normala*) a relatiei. Exista mai multe tipuri de dependente de date: dependente functionale, dependente multivalorice si dependente de jonctiune.

Normalizarea relatiilor consta în transformarea lor (în general prin descompunere), astfel încât relatiile rezultate sa îndeplineasca conditii din ce în ce mai restrictive în ceea ce privesc dependentele de date, adica sa corespunda unor forme normale cât mai avansate.

Prin normalizare se elimina (sau se micsoreaza) redundanta datelor memorate în relatii si anomaliile care provin din aceasta redundanta. Totusi, dupa normalizare, o parte din interogari vor necesita jonctiuni între relatiile rezultate prin descompunere, jonctiuni care nu erau necesare daca relatia nu ar fi fost descompusa si care conduc la cresterea timpilor de executie a unor operatii de interogare. De aceea, în practica proiectarii bazelor de date, normalizarea relatiilor nu se face întotdeauna pâna în forma normala cea mai înalta. Proiectantii pot pastra relatiile într-o forma de normalizare mai scazuta cu conditia ca aceste situatii sa fie bine cunoscute si documentate, pentru a se trata în mod corespunzator operatiile în care ar putea sa apara anomalii de actualizare a relatiilor.

5.1 DEPENDENTE FUNCTIONALE

O dependenta functionala - DF - (*functional dependency*) în relatia cu schema $R = \{A_1, A_2, \ldots A_n\}$ între doua multimi de atribute X si Y (care sunt submultimi ale lui R) exista daca si numai daca, în orice stare a relatiei R, fiecarei valori a atributului (simplu sau compus) X îi corespunde o singura valoare a atributului (simplu sau compus) Y.

O dependenta functionala este deci o constrângere între doua submultimi de atribute X si Y ale unei relatii si se noteaza $X \rightarrow Y$. Ca exprimare, se mai spune ca exista o dependenta functionala de la X la Y, sau ca atributul Y este dependent functional de X. Dependentele functionale se stabilesc lde proiectant la definirea relatiilor pe baza semnificatiei atributelor, astfel încât relatia sa reflecte cât mai corect realitatea pe care o modeleaza.

Tipuri de dependente functionale. Dependentele functionale pot fi *partiale* sau *totale*. O dependenta functionala $X \rightarrow Y$ este partiala daca exista o submultime proprie Z a multimii X (adica $Z \subset X$) care determina functional pe Y ($Z \rightarrow Y$). O dependenta functionala $X \rightarrow Y$ este totala, daca nu exista nici o submultime proprie Z a lui X care sa determine functional pe Y. Din aceasta definitie rezulta ca, daca atributul X este simplu, dependenta functionala $X \rightarrow Y$ este totala.

Dependenta functionala $X \rightarrow Y$ (unde $X \subseteq R$, $Y \subseteq R$) este satisfacuta de orice relatie R daca $Y \subseteq X$. O astfel de dependenta functionala se numeste dependenta functionala *triviala*. Atributele care apartin unei chei se numesc *atribute prime*, iar celelalte se numesc *atribute neprime*.

Proprietatea de a determina functional multimea de atribute ale relatiei R o are si orice cheie CK (primara sau secundara) a relatiei date ($CK \rightarrow R$), dat fiind ca într-o relatie nu pot exista doua tupluri cu aceeasi valoare a cheii. Daca atributul CK este o cheie candidata a relatiei R, atunci, conform proprietatii de ireductibilitate a cheii candidate, dependenta $CK \rightarrow R$ este totala, adica nu exista o submultime proprie CK' a lui CK care sa prezinte proprietatea $CK' \rightarrow R$.

Fiind data o relatie si o multime de dependente functionale care trebuie sa fie satisfacute de orice stare a relatiei, o parte din dependentele functionale pot fi determinate de chei ale relatiei (si acestea sunt constrângeri implicite, care nu produc redundanta datelor si nici anomalii de actualizare a relatiei si sunt impuse automat de sistemul de gestiune), iar alta parte pot fi determinate de alte atribute care nu sunt chei ale relatiei (si acestea sunt constrângeri explicite care produc redundanta datelor si anomalii de actualizare a relatiei).

Dependentele functionale în care atributul determinant nu este o cheie a relatiei nu sunt verificate si nici impuse de sistemul de gestiune si verificarea si impunerea lor se poate face numai procedural, prin triggere, proceduri stocate, sau functii incluse în programele de aplicatie.

Multimi de dependente functionale. În mod obisnuit, proiectantul bazei de date specifica acele dependente functionale între atribute care sunt evidente din punct de vedere semantic. În afara acestor dependente functionale, mai exista un numar destul de mare de dependente functionale care se mentin pentru orice stare a relatiei, si care se pot deduce din multimea celor specificate, folosind regulile de deducere (inferenta - *inference rules*) ale lui Armstrong (reflexivitatea, augmentarea si tranzitivitatea).

Închiderea unei multimi de dependente functionale. Fiind data o multime `F` de dependente functionale, multimea tuturor dependentelor functionale care sunt implicate de `F` se numeste *închiderea multimii* `F` si se noteaza F^+. Multimea tuturor dependentelor functionale se poate deduce prin aplicarea repetata a regulilor de inferenta asupra dependentelor functionale din `F`.

Descompunerea relatiilor. Descompunerea unei relatii, efectuata în scopul normalizarii, nu poate fi facuta oricum, ci trebuie sa fie respectate anumite conditii de conservare a semnificatiei atributelor si a dependentelor dintre ele, asa cum erau definite în relatia initiala. Dintre toate descompunerile posibile, numai o parte au proprietatea ca pot reconstitui prin jonctiune naturala relatia initiala `R`. Fiind data o relatie cu schema `R` si multimea `F` a dependentelor functionale ale acesteia, o descompunere `D` a relatiei `R` se numeste *descompunere reversibila* daca are proprietatile de *jonctiune fara pierdere de informatie* si de *conservare a dependentelor functionale*. Aceste proprietati sunt studiate la cursul de baze de date.

5.2 FORME NORMALE ALE RELATIILOR

Initial, E.F. Codd a propus trei forme normale, numite prima forma (FN1), a doua forma (FN2) si a treia forma normala (FN3). Ulterior, a fost introdusa o definitie mai completa a celei de-a treia forme, care a primit numele de forma normala Boyce-Codd (FNBC). Toate aceste forme normale se refera la conditiile pe care trebuie sa le îndeplineasca *dependentele functionale* între atributele unei relatii. Forme normale superioare, cum sunt a patra forma normala (FN4) si a cincea forma normala (FN5) impun conditii *dependentelor multivalorice*, respectiv *dependentelor de jonctiune* între atributele unei relatii.

O relatie este normalizata în prima forma normala (FN1) *daca fiecare atribut ia numai valori atomice si scalare din domeniul sau de definitie.*

Caracterul de atomicitate se refera la faptul ca valoarea unui atribut are semnificatie numai în ansamblul ei si nu permite descompunerea în componente care sa poata fi manevrate separat. Chiar daca tipul de date peste care este definit domeniul unui atribut este reprezentat prin mai multe componente, acestea nu au semnificatie luate individual. Valoarea scalara a unui atribut se refera la faptul ca un atribut nu poate avea decât o valoare pentru fiecare tuplu. Sistemele SGBD relationale nu admit relatii care sa nu fie cel putin în prima forma normala, dar proiectarea relatiilor normalizate în prima forma normala este simpla si întotdeauna posibila, si a fost deja prezentata în Capitolul 2.

O relatie este normalizata în a doua forma normala (FN2) *în raport cu o multime de dependente functionale `F`, daca este în FN1 si daca în F^+ nu exista nici o dependenta functionala partiala a unui atribut neprim fata de o cheie a relatiei.*

Ca exemplu de normalizare în FN2, se va studia relatia `AP` cu schema `AP(IdAngajat, Nume,Prenume,Adresa,IdProiect,Ore)` si cu multimea dependentelor functionale F_{AP}:

F_{AP} = {IdAngajat→Nume, IdAngajat→Prenume,
IdAngajat→Adresa, {IdAngajat,IdProiect}→Ore}

Cheia primara a relatiei este `{IdAngajat,IdProiect}` si se poate deduce din multimea dependentelor functionale. Se presupune ca toate atributele iau valori atomice si scalare, deci relatia este în FN1. Atributele prime sunt `IdAngajat, IdProiect` iar toate celelalte atribute sunt neprime.

Se observa ca dependenta functionala {IdAngajat,IdProiect}→Nume este partiala, datorita faptului ca submultimea {IdAngajat} a atributului compus cheie primara determina functional atributul Nume: IdAngajat→Nume. Rezulta ca relatia AP nu este în FN2. Într-o astfel de relatie exista date redundante (de exemplu, numele, prenumele si adresa unui angajat se înregistreaza pentru fiecare proiect la care lucreaza angajatul) care produc anomalii de actualizare.

Relatia AP se poate descompune în relatiile A(IdAngajat,Nume,Prenume,Adresa) si P(IdAngajat,IdProiect,Ore), care sunt în FN2 (de fapt, sunt chiar în FNBC, se poate verifica usor acest lucru) si nu prezinta redundanta a datelor si anomalii de actualizare a tuplurilor.

O relatie este normalizata în a treia forma normala (FN3) *în raport cu o multime de dependente functionale F daca este în FN2 si daca în F^+ nu exista nici o dependenta functionala netriviala a unui atribut neprim fata de alt atribut neprim al relatiei.*

Orice relatie formata din doua atribute este în FN3 deoarece ea se afla în FN2 (s-a demonstrat în sectiunea precedenta), si nu poate exista nici un atribut neprim care sa determine functional un alt atribut neprim, deoarece o relatie cu doua atribute nu poate avea decât cel mult un atribut neprim.

Ca exemplu de normalizare a unei relatii în a treia forma normala, se considera schema relatiei AFS(IdAngajat, Nume, Prenume, Adresa, Functie, Salariu), cu multimea dependentelor functionale F_{AFS} = {IdAngajat→Nume, IdAngajat→Prenume, IdAngajat→Functie,Functie→Salariu}. Cheia primara a relatiei este atributul IdAngajat, si ea poate fi dedusa din multimea F_{AFS} a dependentelor functionale.

Se considera ca fiecare atribut ia numai valori atomice si scalare, deci relatia este în FN1. Primele patru dependente functionale sunt dependentele functionale totale ale unor atribute neprime fata de cheia primara a relatiei, deci relatia este în FN2.

Dependenta functionala (Functie→Salariu) semnifica faptul ca în institutia respectiva toti salariatii cu aceeasi functie au acelasi salariu (adica functia determina salariul, ceea ce este plauzibil). Aceasta dependenta functionala a atributului neprim Salariu fata de alt atribut neprim (Functie), arata ca relatia nu este în a treia forma normala (FN3).

Chiar daca relatia AFS este în FN2, în aceasta înca mai exista redundanta a datelor, deoarece valoarea salariului corespunzator unei functii se înregistreaza de mai multe ori, pentru fiecare salariat care detine acea functie. Astfel de redundante si anomaliile de actualizare pe care le provoaca se pot elimina daca se descompune relatia în doua (sau mai multe) relatii, care sa nu contina date redundante.

Relatia AFS se poate descompune în relatiile AF(IdAngajat,Nume,Prenume,Adresa, Functie) si FS(Functie,Salariu). Se poate verifica usor ca relatiile rezultate sunt în FN3.

O relatie cu schema R este în forma normala Boyce-Codd (FNBC) *în raport cu o multime F de dependente functionale daca este în FN1 si daca, pentru orice dependenta functionala netriviala $X \to Y$ din F^+, X este o cheie a relatiei R.*

Se poate observa asemanarea dintre formele normale FN3 si FNBC, ambele impunând conditia ca atributul care determina functional alte atribute sa fie o cheie a relatiei. Forma normala Boyce-Codd este mai restrictiva decât FN3, deoarece în FNBC se impune aceasta conditie tuturor atributelor, prime sau neprime, pe câta vreme în FN3 conditia se impune numai atributelor neprime. Este evident faptul ca o relatie în FNBC este, de asemenea în FN3, dar o relatie în FN3 poate sa fie sau nu în FNBC.

Ca exemplu de normalizare în FNBC, se considera relatia EDP(IdElev,IdDisciplina, IdProfesor), cu cheia candidata {IdElev,IdDisciplina} si cu multimea F_{EDP} a dependentelor functionale:

F_{EDP} = {{IdElev,IdDisciplina}→IdProfesor,IdProfesor→IdDisciplina}

Aceste dependente functionale semnifica o anumita organizare a activitatii de instruire a elevilor: (a) fiecare elev are un singur profesor la o disciplina (de regula) si (b) un profesor preda o singura disciplina (plauzibil).

Se considera ca atributele iau valori atomice si scalare, deci relatia este în FN1. Din multimea dependentelor functionale F_{EDP} se observa ca nu exista dependente functionale partiale fata de cheia relatiei (deci relatia este în FN2) si nu exista nici o dependenta functionala a unui atribut neprim fata de un alt atribut neprim, deci relatia EDP este în FN3. Totusi, relatia EDP nu este în FNBC, datorita

dependentei functionale a atributului prim IdDisciplina fata de atributul neprim IdProfesor. O relatie care nu este în FNBC prezinta, ca orice relatie incomplet normalizata, redundanta a datelor si anomalii de actualizare. De exemplu, în relatia EDP, pot exista mai multe tupluri care contin o anumita valoare a atributului IdProfesor (deoarece mai multi elevi studiaza cu acelasi profesor) si, de fiecare data, este înregistrata disciplina (atributul IdDisciplina) pe care o preda profesorul respectiv. Din aceasta redundanta a datelor rezulta mai multe anomalii de inserare, stergere si actualizare a tuplurilor.

Normalizarea relatiei EDP astfel încât relatiile obtinute sa fie în FNBC se poate face prin descompunerea acesteia. Se pot încerca trei descompuneri:

D_1 = {EP,PD}: EP ={IdElev,IdProfesor},PD = {IdProfesor,IdDisciplina};
D_2 = {ED,PD}: ED = {IdElev,IdDisciplina},PD = {IdProfesor,IdDisciplina};
D_3 = {EP,ED}: EP = {IdElev,IdProfesor}, ED = {IdElev,IdDisciplina}.

Se observa ca relatiile rezultate în oricare din aceste descompuneri sunt relatii în FNBC (fiind relatii formate din doua atribute) si ca, în oricare din aceste descompuneri, se pierde dependenta functionala {IdElev,IdDisciplina}→IdProfesor. Rezulta ca relatia EDP nu poate fi descompusa în mod reversibil în relatii FNBC si ca, pentru respectarea tuturor constrângerilor dintr-o relatie ca relatia EDP, sunt necesare masuri speciale.

5.3 VERIFICAREA SI IMPUNEREA PROGRAMATICA A CONSTRÂNGERILOR EXPLICITE

Analiza normalizarii relatiilor trebuie sa fie realizata pentru orice proiect de baze de date, pentru a asigura functionarea corecta a acesteia. Dupa ce se decide forma normala a unei relatii, este necesar sa se prevada procedurile de verificare a tuturor constrângerilor explicite ramase: fie dependente de date care nu sunt determinate de chei ale relatiei (într-o relatie cu o forma de normalizare scazuta), fie dependentele functionale pierdute prin descompunerea relatiei, care devin constrângeri explicite între relatiile rezultate prin descompunere.

5.3.1 IMPUNEREA DEPENDENTELOR FUNCTIONALE CARE NU SUNT DETERMINATE DE CHEILE RELATIILOR

Daca o relatie se pastreaza într-o forma de normalizare mai redusa, atunci trebuie sa se prevada proceduri de verificare si impunere a dependentelor de date care nu sunt determinate de chei ale relatiei (constrângeri explicite).

5.3.1.1 Verificarea si impunerea dependentelor functionale în relatia AP

Daca relatia AP nu se normalizeaza (nu se descompune) atunci trebuie sa se prevada proceduri speciale care sa verifice si sa impuna dependentele care nu sunt determinate cheia relatiei. Pentru aceasta este suficient ca la orice operatie de actualizare a relatiei sa se verifice valorile care urmeaza sa fie introduse (sau modificate) si sa nu se admita doi sau mai multi angajati cu acelasi numar de identificare (IdAngajat) dar cu nume, prenume sau adresa diferite.

În continuare sunt prezentate trei dintre modalitatile posibile de verificare si impunere a dependentelor functionale care nu sunt determinate de chei: printr-un trigger în limbajul Transact-SQL, printr-o procedura stocata Transact-SQL si printr-o functie definita într-un program de aplicatie.

Verificarea si impunerea DF în relatia AP printr-un trigger Transact-SQL. În limbajul Transact-SQL un trigger se poate defini cu comanda CREATE TRIGGER care are urmatoarea sintaxa generala:

```
CREATE TRIGGER nume_trigger ON nume_tabel
   {FOR|AFTER|INSTEAD OF}{[DELETE][,INSERT][,UPDATE]}
   AS instructiuni
```

Un trigger se poate defini pentru oricare din comenzile DELETE, INSERT, UPDATE sau pentru orice combinatie a acestora (scrise în orice ordine, separate prin virgula) si consta din toate instructiunile Transact-SQL care urmeaza dupa cuvântul cheie AS. Cele trei optiuni FOR, AFTER,

INSTED OF permit crearea de fapt a doar doua tipuri de triggere, optiunile FOR si AFTER fiind echivalente. Un trigger cu optiunea FOR sau AFTER declanseaza executia instructiunilor proprii dupa ce operatiile prevazute de instructiunea SQL asociata (INSERT, UPDATE, DELETE) au fost executate si numai daca toate constrângerile prevazute de aceste instructiuni au fost verificate cu succes. Un trigger cu optiunea INSTEAD OF declanseaza anumite actiuni care se efectueaza în locul celor prevazute de instructiunea SQL asociata.

În Programul 5.1 se creeaza un trigger pentru tabelul AP de tip FOR (AFTER) pentru operatia INSERT, care se declanseaza dupa orice inserare reusita a unei linii în tabelul respectiv. Sistemul de gestiune face toate verificarile posibile asupra conditiilor pe care le cunoaste: unicitatea cheilor, eventuale verificari CHECK, sau chei straine (daca exista), însa va considera acceptabila si va efectua introducerea unei linii care violeaza o dependenta functionala care nu este determinata de cheia relatiei. Triggerul trebuie sa verifice aceasta situatie si sa o corecteze (sa stearga linia respectiva).

Trigger Transact-SQL pentru verificarea si impunerea unor DF în relatia AP (Programul 5.1).

```
CREATE TRIGGER tg_AP ON dbo.AP
FOR INSERT AS
DECLARE @v_angajat int, @v_nume varchar(20)
DECLARE @v_prenume varchar(20), @v_adresa varchar(20)
DECLARE @n_angajat int, @n_proiect varchar(20)
DECLARE @n_nume varchar(20), @n_prenume varchar(20)
DECLARE @n_adresa varchar(20)
-- Memorarea valorilor atributelor liniei nou inserate
SELECT @n_angajat = IdAngajat, @n_proiect = IdProiect,
      @n_nume = Nume, @n_prenume = Prenume,@n_adresa = Adresa
      FROM INSERTED
-- Crearea unui cursor
DECLARE cursor_AP CURSOR FOR
      SELECT IdAngajat,Nume,Prenume,Adresa FROM AP
OPEN cursor_AP
-- Parcurgerea liniilor tabelului AP folosind cursorul
FETCH cursor_AP INTO @v_angajat, @v_nume, @v_prenume, @v_adresa
WHILE (@@FETCH_STATUS = 0)
BEGIN
      IF @n_angajat=@v_angajat AND(@n_nume!=@v_nume
         OR @n_prenume!=@v_prenume OR @n_adresa!=@v_adresa)
         BEGIN
         -- Daca nu este satisfacuta DF,se sterge linia nou introdusa
         DELETE FROM AP WHERE IdAngajat=@n_angajat AND IdProiect=@n_proiect
         BREAK
         END
      FETCH cursor_AP INTO @v_angajat, @v_nume, @v_prenume, @v_adresa
END
CLOSE cursor_AP
DEALLOCATE cursor_AP
```

Prima operatie care se efectueaza în trigger este de a memora în variabilele locale ale programului (n_angajat, n_proiect, n_nume, n_prenume, n_adresa) valorile atributelor liniei nou introduse, prin selectarea lor din tabelul temporar INSERTED, creat de trigger. Atunci când se declanseaza un trigger, sistemul creaza în memorie doua tabele temporare cu numele INSERTED si DELETED, cu aceeasi structura ca tabelul pe care este definit triggerul. În tabelul DELETED se memoreaza linia (sau liniile) actualizate sau sterse, care contin vechile valori ale atributelor, asa cum erau înainte de o comanda DELETE sau UPDATE. În tabelul INSERTED se memoreaza liniile cu noile valori ale atributelor care se introduc sau se actualizeaza în cursul operatiilor de INSERT si UPDATE. Aceste tabele temporare sunt folosite în trigger pentru a se efectua comparatii între vechile valori si valorile modificate si se pot efectua anumite actiuni în functie de rezultatul acestor comparatii.

În continuare se parcurg toate liniile tabelului AP si se compara atributele IdAngajat, Nume, Prenume, Adresa ale liniei nou introduse cu cele ale tuturor liniilor existente în tabel. Daca exista doua linii care au aceeasi valoare a atributului IdAngajat, dar valori diferite ale unuia din atributele Nume,Prenume sau Adresa, atunci nu se admite noua linie inserata; de fapt se sterge acea linie, deoarece ea fusese deja înscrisa în tabel.

Pentru aceasta operatie se creaza un cursor (cursor_AP) si se extrag pe rând liniile (cu operatii FETCH). Pentru fiecare linie extrasa se verifica conditia descrisa si, daca nu este îndeplinita, se sterge linia nou introdusa si se termina executia triggerului (dupa închiderea si dezalocarea cursorului).

În sistemul SQL Server un triger se poate memora în baza de date fie prin executia programului acestuia (asa cum este fisierul script Program_5_1.sql din acest director) cu ajutorul utilitarului *osql* sau *Query Analyzer*, fie folosind *SQL Server Enterprise Manager*. În *Enterprise Manager* se foloseste comanda *All Tasks -> Manage Triggers*, care se poate actiona din meniul de context care se deschide prin apasarea butonului dreapta al mouse-ului atunci cînd este selectat tabelul pentru care se defineste triggerul. La actionarea acestei comenzi se deschide o fereastra (*Trigger Properties*), iar din caseta *Name* a acestei ferestre se poate selecta oricare trigger al tabelului, sau se poate crea un trigger nou. Atunci cînd se creeaza un trigger, în ferestra de editare se ofera un prototip al instructiunii de definire a triggerului, care poate fi modificat asa cum este necesar. Atât triggerele nou create cât si cele existente si modificate, se pot verifica din punct de vedere sintactic (cu comanda *Check Syntax*) si se memoreaza în baza de date (cu comanda *Apply*).

Verificarea si impunerea DF în relatia AP printr-o procedura stocata Transact-SQL. Pentru a crea o procedura stocata Transact-SQL se foloseste urmatoarea constructie:

```
CREATE PROC[EDURE] nume_procedura [optiuni]AS instructiuni
```

O procedura stocata poate fi stearsa cu instructiunea:

```
DROP PROCEDURE nume_procedura;
```

De asemenea, este posibila modificarea unei proceduri prin comanda ALTER PROCEDURE. Dupa ce a fost creata, o procedura stocata poate fi apelata dintr-un lot de executie, dintr-un declansator sau dintr-o alta procedura stocata (executie imbricata) cu comanda:

```
EXEC[UTE] nume_procedura [parametri_apel]
```

Procedurile stocate pot sa primeasca parametri de intrare si pot returna valori prin intermediul parametrilor de iesire. Fiecare parametru al unei proceduri stocate se declara ca optiune în instructiunea de declarare a procedurii folosind sintaxa urmatoare:

```
@nume_parametru tip_date[VARYING][= val_implicita][OUTPUT]
```

Parametrii sunt variabile locale la nivelul procedurii; numele fiecarui parametru începe cu caracterul @ (ca si numele oricarei alte variabile locale) si pentru fiecare parametru trebuie sa fie declarat tipul de date. Optiunea OUTPUT indica faptul ca parametrul respectiv este un parametru de iesire, prin intermediul caruia procedura returneaza date programului apelant. În lipsa acestei optiuni, parametrul este de intrare si la apelul procedurii trebuie sa fie precizata valoarea acestuia.

Modul de verificare a dependentelor functionale care nu sunt determinate de chei în relatia AP folosind o procedura stocata este dat în Programul 5.6.

Fisierul *Program_5_6a.sql* contine scriptul de creare a procedurii stocate. La executia acestui program se creeaza procedura stocata sp_normalizare în baza de date curenta.

Acestei proceduri i se transmit ca parametri de intrare valorile atributelor tuplului care urmeaza sa fie inserat si ea returneaza (în parametrul de iesire rezultat) valoarea 1 daca datele au fost corecte din punct de vedere al respectarii dependentelor functionale si s-a apelat instructiunea INSERT si valoarea 0 daca datele respective violeaza una din dependentele functionale care se testeaza si nu s-au introdus.

Ca si triggerul prezentat mai înainte, procedura foloseste un cursor pentru parcurgerea tuturor liniilor tabelului AP si compararea noilor valori (care urmeaza sa fie introduse) cu cele existente deja în tabel.

Procedura stocata Transact-SQL pentru verificarea si impunerea unor DF în relatia AP (Program 5.6a)

```
-- Stergerea obiectului cu acelasi nume, daca exista
IF EXISTS (SELECT * FROM dbo.sysobjects
      WHERE name = 'sp_normalizare')
DROP PROCEDURE sp_normalizare
GO
-- Crearea procedurii stocate
CREATE PROCEDURE sp_normalizare
         @n_angajat INT, @n_nume varchar(20),@n_prenume varchar(20),
         @n_adresa varchar(20), @n_proiect varchar(10), @n_ore int,
         @raspuns INT OUTPUT
AS
DECLARE @v_angajat int, @v_nume varchar(20), @v_prenume varchar(20)
DECLARE @v_adresa varchar(20), @v_proiect varchar(10), @v_ore int
DECLARE cursor_AP CURSOR FOR SELECT IdAngajat,Nume,Prenume,Adresa FROM AP
OPEN cursor_AP
SELECT @raspuns = 1
FETCH cursor_AP INTO @v_angajat, @v_nume, @v_prenume, @v_adresa
WHILE (@@FETCH_STATUS = 0)
BEGIN
      IF @n_angajat = @v_angajat AND(@n_nume != @v_nume
            OR @n_prenume != @v_prenume OR @n_adresa != @v_adresa)
         BEGIN
            -- Nu este satisfacuta DF
            SELECT @raspuns = 0
            BREAK
         END
         FETCH cursor_AP INTO @v_angajat, @v_nume, @v_prenume, @v_adresa
END
CLOSE cursor_AP
DEALLOCATE cursor_AP
IF @raspuns = 1
      INSERT INTO AP VALUES(@n_angajat, @n_nume, @n_prenume,
          @n_adresa,@n_proiect,@n_ore)
GO
```

În programele de aplicatii se va înlocui operatia de inserare (transmiterea catre SGBD a instructiunii SQL `INSERT`) cu apelul acestei proceduri stocate, care verifica datele înainte de a le insera, astfel:

Apelul procedurii stocate sp_normalizare(Program 5.6 –b)

```
DECLARE @rezultat int
EXECUTE sp_normalizare @n_angajat = 1,@n_nume = 'Nume1',
      @n_prenume = 'PrenumeX',@n_adresa = 'Adresa1',
      @n_proiect = 'P5', @n_ore =500,@raspuns = @rezultat OUTPUT
IF @rezultat = 1 PRINT 'Introducere corecta'
ELSE PRINT 'Nu s-a respectat DF - nu s-a introdus nimic'
GO
```

Verificarea si impunerea DF în relatia AP printr-o functie în programul de aplicatie. Programul 5.2 (care este dat în directorul acestui capitol al îndrumarului) este varianta ODBC a programelor precedente: se verifica valorile care urmeaza sa fie inserate în tabelul `AP` si se accepta numai acelea care satisfac toate dependentele functionale.

5.3.1.2 Verificarea si impunerea dependentelor functionale în relatia AFS

Daca nu se normalizeaza relatia `AFS`, atunci trebuie sa fie prevazuta o procedura care sa verifice si sa impuna dependenta functionala `Functie→Salariu`, care nu este determinata de cheia relatiei. În continuare sunt prezentate câteva dintre solutiile posibile ale acestei probleme: un trigger

Transact-SQL (pentru sistemul SQL Server), un trigger si o procedura stocata PL/SQL (pentru sistemul Oracle). Se considera ca atributul cheie primara `IdAngajat` nu este de tip `IDENTITY` (în SQL Server) sau nu se foloseste o secventa (în Oracle).

Verificarea si impunerea DF în relatia AFS printr-un trigger Transact-SQL. În Programul 5.3 este prezentat codul Transact-SQL al unui trigger care impune aceasta dependenta functionala. Acesta are o functionare asemanatoare cu cea a triggerului din Programul 5.1.

Verificarea si impunerea DF în relatia AFS printr-un trigger PL/SQL. În PL/SQL un trigger se defineste în modul urmator:

```
CREATE [OR REPLACE] TRIGGER nume_trigger
{BEFORE | AFTER | INSTEAD OF} lista_operatii [FOR EACH ROW]
{bloc PL/SQL};
```

Se pot crea trei tipuri de triggere (`BEFORE`, `AFTER` si `INSTEAD OF`), pentru una sau mai multe operatii (`INSERT`, `UPDATE`, `DELETE`) si exista mai multe conditionari între tipul triggerului, tipul operatiei si optiunea `FOR EACH ROW`. Un trigger `INSTEAD OF` nu se poate crea decât pe o vedere (nu pe un tabel), dar aceasta conditie nu complica prea mult lucrurile, deoarece o vedere identica cu un tabel se poate crea foarte simplu.

În majoritatea cazurilor, din blocul PL/SQL al triggerului sunt accesibile doua tipuri de variabile ale triggerului, care se dau cu numele coloanei prefixat cu `:NEW`, respectiv `:OLD`. De exemplu, daca în tabel exista atributul `Nume`, atunci din trigger se pot accesa variabilele `:NEW.Nume` si `:OLD.Nume`, care reprezinta valoarea atributului înainte si, respectiv, dupa inserare (modificare).

Programul 5.7 dat în continuare este un trigger PL/SQL pentru verificarea dependentei functionale `Functie->Salariu` în vederea `AF` care este identica cu tabelul `AFS` (este definita cu instructiunea: `CREATE VIEW AF AS (SELECT * FROM AFS)`).

Trigger PL/SQL pentru verificarea si impunerea DF în relatia `AFS` (Programul 5.7).

```
SET SERVEROUTPUT ON
CREATE OR REPLACE TRIGGER tg_AF INSTEAD OF INSERT ON AF
DECLARE
      m_functie varchar2(20); m_salariu decimal; acceptare number;
      CURSOR cursor_AF IS SELECT Functie, Salariu FROM AF;
BEGIN
      DBMS_OUTPUT.PUT_LINE('Trigger AF');
      acceptare := 1;
      OPEN cursor_AF;
      LOOP
          FETCH cursor_AF INTO m_functie, m_salariu;
          IF cursor_AF%NOTFOUND THEN EXIT;
          ELSE
             IF :NEW.Functie = m_functie AND :NEW.Salariu != m_salariu
             THEN
                acceptare := 0; EXIT;
             END IF;
          END IF;
       END LOOP;
       CLOSE cursor_AF;
       IF acceptare = 1
          THEN
            INSERT INTO AFS VALUES (:NEW.IdAngajat, :NEW.Nume,
             :NEW.Prenume, :NEW.DataNasterii, :NEW.Adresa,
             :NEW.Functie, :NEW.Salariu);
       END IF;
END;
```

În trigger se parcurg liniile vederii `AF` folosind un cursor, si pentru fiecare linie extrasa din cursor se compara valorile atributelor `Functie`, `Salariu` cu valorile care urmeaza sa fie introduse

(:NEW.Functie si :NEW.Salariu) si se pozitioneaza o variabila locala (acceptare) în functie de rezultatul acestei comparatii (la valoarea 1 daca se respecta dependenta functionala si la valoarea 0 daca nu se respecta). Daca, dupa parcurgerea tuturor liniilor vederii, variabila acceptare este 1, atunci se insereaza datele dorite în tabelul AFS, altfel nu se executa nimic.

Triggerul este declansat la orice operatie de inserare în vederea AF, ceea ce înseamna ca în programele de aplicatii trebuie sa se faca inserarile în vederea AF, atunci când se doreste introducerea datelor în tabelul AFS.

Verificarea si impunerea DF în relatia AFS printr-o procedura stocata PL/SQL. În limbajul PL/SQL o procedura stocata se defineste în felul urmator:

```
CREATE [OR REPLACE] nume_procedura [(lista_parametri)]
   {AS | IS}
   [declaratii_locale]
BEGIN
   instructiuni
[EXCEPTION  functie_prelucrare_exceptie]
END;
```

Parametrii din lista de parametri sunt separati prin virgula si pot fi de trei tipuri: de intrare, de iesire, sau de intrare-iesire. Fiecare parametru se specifica prin numele lui, tipul (IN, OUT sau IN OUT) si tipul variabilei (number, varchar, etc.).

Programul 5.8-a prezentat în continuare este o procedura stocata PL/SQL care asigura verificarea si impunerea dependentei functionale Functie→Salariu în relatia AFS.

Procedura stocata PL/SQL pentru verificarea DF în relatia AFS (Programul 5.8-a)

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE sp_AFS (
      n_nume IN varchar2, n_prenume IN varchar2, n_adresa IN varchar2,
      n_functie IN varchar2, n_salariu IN decimal, raspuns OUT number)
AS
      v_functie varchar2(20); v_salariu decimal;
      CURSOR cursor_AFS IS SELECT Functie, Salariu FROM AFS;
BEGIN
      raspuns := 1;
      OPEN cursor_AFS;
      LOOP
           FETCH cursor_AFS INTO v_functie, v_salariu;
           IF cursor_AFS%NOTFOUND THEN EXIT;
           ELSE
              IF n_functie = v_functie AND n_salariu != v_salariu
                  THEN  raspuns := 0; EXIT;
              END IF;
           END IF;
      END LOOP;
      CLOSE cursor_AFS;
      IF raspuns = 1
         THEN
            INSERT INTO AFS VALUES(n_angajat, n_nume, n_prenume,
               n_adresa, n_functie, n_salariu);
      END IF;
END;
```

Procedura stocata sp_AFS primeste ca argumente valorile atributelor tuplului care trebuie sa fie inserat, si verifica daca aceste atribute respecta dependenta functionala Functie→Salariu folosind un cursor. Daca dependenta functionala este respectata, atunci se executa inserarea liniei, altfel nu. În aceslasi timp depune în parametrul de raspuns (de tip OUT) valoarea 1, daca s-a efectuat inserarea, sau valoarea 0, daca nu este respectata dependenta dorita si nu s-a efectuat inserarea.

Acesta procedura stocata se poate apela din orice program în locul inserarii (cu instructiunea SQL `INSERT`) a unei linii în tabelul AFS. Modul de apel al acestei proceduri este dat în continuare (Programul 5.8-b).

Apelul procedurii stocate `sp_AFS` (Programul 5.8-b).

```
DECLARE
      rezultat number;
BEGIN
       sp_AFS(4,'Nume4','Prenume4','Adresa4','Functie1',40000,rezultat);
       IF rezultat = 1
            THEN DBMS_OUTPUT.PUT_LINE('Introducere corecta');
            ELSE DBMS_OUTPUT.PUT_LINE('Nu s-a respectat DF,
                                  nu s-a introdus nimic');
       END IF;
END;
```

5.3.1.3 Impunerea dependentelor functionale în relatia EDP

Daca relatia `EDP` nu se descompune, atunci trebuie sa fie prevazuta o procedura pentru verificarea si impunerea dependentei functionale `IdProfesor`→`IdDisciplina`, care nu este determinata de cheia relatiei. Aceasta procedura poate fi un trigger definit pe relatia `EDP` (asa cum este Programul 5.4, Transact-SQL), o procedura stocata sau o functie într-un program de aplicatie.

5.3.2 IMPUNEREA CONSTRÂNGERILOR PIERDUTE ÎN CURSUL DESCOMPUNERII RELATIILOR

Daca prin descopmunerea unei relatii se pierde o dependenta functionala, aceasta poate fi impusa explicit printr-o procedura stocata sau o functie în programul de aplicatie, care executa jonctiunea între relatiile rezultate si impune constrângerea respectiva.

De exemplu, la descompunerea relatiei `EDP` în relatiile `EP` si `PD` s-a pierdut dependenta functionala {`IdElev`,`IdDisciplina`}→`IdProfesor`, si constrângerea respectiva poate fi impusa în mod explicit printr-o procedura stocata sau o functie în programul de aplicatie.

În Programul 5.5 (care se gaseste în directorul acestui capitol al îndrumarului) este prezentata o functie de testare care este apelata în programul de aplicatie (dezvoltat pe baza interfetei ODBC) ori de câte ori se introduc valori noi ale atributelor `IdElev`,`IdDisciplina`,`IdProfesor` în relatiile `EP` si `PD`. Daca se respecta constrângerea de mai sus, atunci se executa doua instructiuni `INSERT`, în tabelul `EP` (pentru valorile `nIdElev` si `nIdProfesor`) si în tabelul `PD` (pentru valorile `nIdProfesor`, `nIdDisciplina`). Daca nu se respecta, atunci nu se insereaza date în nici unul dintre tabele.

Exercitii - Capitolul 5

5.1 În baza de date proprie din SQL Server sau Oracle adaugati tabelul AP cu urmatoarea schema:

AP(IdAngajat,Nume,Prenume,Adresa,IdProiect,NrOre)

În acest tabel introduceti urmatoarele linii de date:

```
(1, 'Nume1', 'Prenume1', 'Adresa1', 'P1',100)
(1, 'Nume1', 'Prenume1', 'Adresa1', 'P2',200)
(2, 'Nume2', 'Prenume2', 'Adresa2', 'P1',300)
(2, 'Nume2', 'Prenume2', 'Adresa2', 'P2',400)
```

Creati triggerul pentru verificarea dependentelor functionale care nu sunt determinate de cheia primara a relatiei: IdAngajat→Nume,IdAngajat→Prenume,IdAngajat→Adresa. În sistemul SQL Server triggerul se poate crea prin executia fisierului script *Program_5_1.sql*. Pentru sistemul Oracle se creaza un triggger cu functionare similara. Dupa crearea triggerului încercati sa introduceti urmatorul tuplu:

```
(1, 'Nume1','PrenumeX','Adresa1', 'P3', 50);
```

Ce mesaj obtineti la executia acestei instructiuni? Care este continutul tabelului AP dupa aceasta executie? Care ar putea fi forma corecta a acestei instructiuni?

5.2 Eliminati triggerul de verificare a dependentelor functionale în relatia AP si creati o procedura stocata pentru aceeasi operatie. În SQL Server procedura stocata se creaza prin executia scriptului Program_5_6a.sql, iar în Oracle prin executia scriptului Program_5_8a.sql. Apelati procedura stocata si verificati executia corecta a operatiei de verificare a dependentelor functionale în relatia AP.

5.3 În baza de date proprie din SQL Server sau Oracle adaugati tabelul AFS cu urmatoarea schema:

AFS(IdAngajat,Nume,Prenume,Adresa,Functie,Salariu)

Se considera ca atributul cheie primara IdAngajat nu este de tip IDENTITY (în SQL Server) sau nu foloseste o secventa (în Oracle). În acest tabel introduceti urmatoarele linii de date:

```
(1, 'Nume1', 'Prenume1', 'Adresa1', 'Functie1', 10000)
(2, 'Nume2', 'Prenume2', 'Adresa2', 'Functie2', 20000)
(3, 'Nume3', 'Prenume3', 'Adresa3', 'Functie3', 30000)
```

Creati triggerul pentru impunerea dependentei functionale Functie→Salariu care nu este determinata de cheia primara a relatiei prin executia fisierului script *Program_5_3.sql* pentru SQL Server sau a fisierului *Program_5_7.sql* pentru Oracle. Dupa crearea triggerului încercati sa introduceti urmatoarele date în tabelul AFS (sau în vederea asociata):

```
(4,'Nume4','Prenume4','Adresa4','Functie3',300);
```

Ce mesaj obtineti la executia acestei instructiuni? Care este continutul tabelului AFS dupa aceasta executie? Care ar putea fi forma corecta a acestei operatii?

5.4 Creati o procedura stocata care sa verifice dependenta functionala Functie→Salariu în tabelul AFS. Pentru sistemul Oracle procedura se creeaza prin executia scriptului din fisierul Program_5_6a.sql. Pentru sistemul SQL Server scrieti si executati o procedura Transact-SQL cu functionare similara. Distrugeti triggerul creat în execrcitiile precedente pentru acest tabel (cu instructiunea DROP TRIGGER) si apelati procedura stocata înainte de inserarea datelor în tabelul AFS. Verificati functionarea corecta a inserarii.

5.5 În baza de date proprie din sistemul SQL Server sau Oracle adaugati tabelul EPD cu urmatoarea schema:

EDP(IdElev,IdDisciplina,IdProfesor).

În acest tabel introduceti urmatoarele linii de date:

```
('E1', 'D1' ,'P1')
('E1', 'D2' ,'P2')
```

```
('E2', 'D1' ,'P1')
('E2', 'D2' ,'P2')
```

Creati triggerul pentru impunerea dependentei functionale IdProfesor→IdDisciplina care nu este determinata de cheia primara a relatiei (IdElev,IdDisciplina). Pentru sistemul SQL Server triggerul se poate crea prin executia fisierului script *Program_5_4.sql.* Pentru sistemul Oracle scrieti si executati un program PL/SQL de creare a unui trigger cu functionare similara. Dupa crearea triggerului încercati sa introduceti urmatoarele date în tabelul EDP:

```
('E3','D3','P1');
```

Ce mesaj obtineti la executia acestei instructiuni? Care este continutul tabelului EDP dupa aceasta executie? Care ar putea fi forma corecta a acestei instructiuni?

5.6* În baza de date proprie din SQL Server sau Oracle definiti relatia MP:

MP(IdMedic,Nume,Prenume,IdPacient,DataConsultatiei,Diagnostic)

În aceasta relatie trebuie sa fie respectate urmatoarele dependente functionale: IdMedic→Nume, IdMedic→Prenume,{IdMedic,IdPacient,DataConsultatiei}→Diagnostic. Dezvoltati o procedura stocata (Transact-SQL sau PL/SQL) care sa verifice si sa impuna dependentele functionale care nu sunt determinate de cheia relatiei (IdMedic→Nume,IdMedic→Prenume). Aceasta procedura trebuie sa testeze valorile care urmeaza sa fie inserate si sa nu admita introducerea unui nou tuplu care are valoarea atributului IdMedic egala cu valoarea acestuia dintr-un tuplu oarecare, dar valori diferite ale unuia din atributele Nume sau Prenume.

Dupa crearea triggerului inserati urmatoarele linii în tabelul MP:

```
(1, 'Nume1', 'Prenume1', 100, '03.04.2003', 'D1')
(1, 'Nume1', 'Prenume1', 200, '03.04.2003', 'D1')
(2, 'Nume2', 'Prenume2', 100, '07.04.2003', 'D2')
```

Verificati impunerea dependentelor functionale care nu sunt determinate de cheia relatiei (IdMedic→Nume,IdMedic→Prenume) încercând sa inserati tuplul (1,'Nume1','PrenumeX', 300,'07.07.2003','D4'). Daca programul functioneaza corect, acest tuplu trebuie sa fie rejectat, dat fiind ca violeaza dependenta functionala IdMedic→Prenume.

5.7* În baza de date proprie din sistemul SQL Server sau Oracle definiti relatia SE:

SE(IdStudent,Nume,Prenume,IdDisciplina,DataExamen,Nota)

În aceasta relatie trebuie sa fie respectate urmatoarele dependente functionale: IdStudent→Nume, IdStudent→Prenume,{IdStudent,IdDisciplina,DataExamen}→Nota. Dezvoltati un trigger (Transact-SQL sau PL/SQL) care sa verifice si sa impuna dependentele functionale care nu sunt determinate de cheia relatiei (IdStudent→Nume,IdStudent→Prenume). Aceast trigger trebuie sa testeze valorile care se insereaza si sa nu admita introducerea unui tuplu care are valoarea atributului IdStdudent egala cu valoarea acestuia dintr-un tuplu oarecare, dar are valori diferite ale unuia din atributele Nume sau Prenume. Inserati urmatoarele linii în tabelul SE:

```
(1, 'Nume1', 'Prenume1', 100, '03.04.2003', 8)
(1, 'Nume1', 'Prenume1', 200, '03.04.2003', 7)
(2, 'Nume2', 'Prenume2', 100, '07.04.2003', 9)
```

Verificati impunerea dependentelor functionale care nu sunt determinate de cheia relatiei (IdStudent→Nume,IdStudent→Prenume) încercând sa inserati tuplul (1,'Nume1', 'PrenumeX', 300,'07.07.2003',10). Daca programul functioneaza corect, acest tuplu trebuie sa fie rejectat, dat fiind ca violeaza dependenta functionala IdStudent→Prenume.