Algebra relaţională

Limbajul de definire a datelor (LDD) precizează entităţile, relaţiile dintre ele, atributele, structura atributelor, cheile, constrângerile, prin urmare defineşte structura obiectelor bazei de date (schema bazei).

Limbajul de prelucrare a datelor (LMD) dintr-o bază de date relaţionale cuprinde aspecte referitoare la introducerea, eliminarea, modificarea şi căutarea datelor.

- **Introducerea** datelor – permite adăugarea de tupluri la o relaţie. Tuplurile pot fi introduse de utilizator sau pot fi obţinute din alte relaţii existente în baza de date.
- **Eliminarea** datelor – permite ştergerea tuplurilor ce satisfac condiţii date.
- **Modificarea** datelor – permite reactualizarea tuplurilor ce satisfac condiţii date cu noi valori ale atributelor sau cu rezultate ale unor operaţii aritmetice efectuate asupra unor valori existente.
- **Căutarea** datelor – permite găsirea tuplurilor sau a unor părţi ale tuplurilor ce satisfac condiţii date.

Modelul relaţional oferă două mulţimi de operatori pe relaţii:

- algebra relaţională (filtrele se obţin aplicând operatori specializaţi asupra uneia sau mai multor relaţii din cadrul bazei relaţionale);
- calculul relaţional (filtrele se obţin cu ajutorul unor formule logice pe care tuplurile rezultatului trebuie să le satisfacă).

Algebra relaţională a fost introdusă de E.F. Codd ca o mulţime de operaţii formale acţionând asupra unor relaţii şi având ca rezultat alte relaţii. Baza teoretică pentru limbajele de interogare relaţionale o constituie operatorii introduşi de Codd pentru prelucrarea relaţiilor.

Operatorii modelului relaţional definesc operaţiile care se pot efectua asupra relaţiilor în scopul realizării funcţiilor de prelucrare asupra BD.

Operatorii sunt numai pentru citire (nu actualizeaza operanzi)!!!

Scopul fundamental al algebrei relationale este de a permite scrierea expresiilor relationale. Expresiile servesc ca o reprezentare de nivel superior, simbolică, a intenţiilor utilizatorului şi pot fi supuse unei diversităţi de reguli de transformare (optimizare).

Relaţiile sunt închise faţă de algebra relaţională (operanzii şi rezultatele sunt relaţii → ieşirea unei operaţii poate deveni intrare pentru alta) → posibilitatea imbricării rxpresiilor în algebra relaţională).

Operatorii algebrei relaţionale sunt:

- operatori tradiţionali pe mulţimi (UNION, INTERSECT, PRODUCT, DIFFERENCE);
- operatori relaţionali speciali (PROJECT, SELECT, JOIN, DIVISION).

Calculul relaţional reprezintă o adaptare a calculului predicatelor la domeniul bazelor de date relaţionale. Ideea de bază este de a identifica o relaţie cu un predicat. Pe baza unor predicate iniţiale, prin aplicarea unor operatori ai calculului cu predicate (conjuncţia, disjuncţia, negaţia, cuantificatorul existenţial şi cel universal) se pot defini noi relaţii.

Calculul relaţional poate să fie orientat pe tupluri sau orientat pe domenii.

Echivalenţa dintre algebra relaţională şi calculul relaţional a fost demonstrată de J.D.Ullman. Această echivalenţă arată că orice relaţie posibil de definit în algebra relaţională poate fi definită şi în cadrul calcului relaţional, şi reciproc.

Operatorii (unari sau binari) algebrei relaţionale realizează următoarele funcţii:

- SELECT (selecţie) – extrage tupluri ce satisfac o condiţie specificată;
- PROJECT (proiecţie) – extrage atributele specificate;
- DIFFERENCE (diferenţă) – extrage tupluri care apar într-o relaţie, dar nu apar în cealaltă;
- PRODUCT (produs cartezian) – generează toate perechile posibile de tupluri, primul element al perechii fiind luat din prima relaţie, iar cel de-al doilea element din cealaltă relaţie;
- UNION (reuniune) – reuneşte două relaţii;
- INTERSECT (intersecţie) – extrage tupluri care apar în ambele relaţii;
- DIVISION (diviziune) – extrage valorile atributelor dintr-o relaţie, care apar în toate valorile atributelor din cealaltă relaţie;
- JOIN (compunere) – extrage tupluri din mai multe relaţii corelate:
- NATURAL JOIN (compunere naturală) – combină tupluri din două relaţii, cu condiţia ca atributele comune să aibă valori identice;

- SEMI-JOIN (semi-compunere) – selectează tupluri ce aparţin unei singure relaţii, care sunt corelate cu tupluri din cea de a doua relaţie;
- Θ-JOIN (Θ-compunere) – combină tupluri din două relaţii (nu neaparat corelate), cu condiţia ca valorile atributelor specificate să satisfacă o anumită condiţie;
- OUTER JOIN (compunere externă) – combină tupluri din două relaţii, astfel încât condiţiile de corelare să fie satisfăcute. Tuplurile din orice relaţie care nu satisfac aceste condiţii sunt completate cu valori *null*.

Pentru operatorii UNION, INTERSECT şi DIFFERENCE, se presupune că sunt aplicaţi numai la relaţii având aceeaşi aritate, iar ordinea (nu numele) atributelor este aceeaşi.

Operatorul PROJECT

Proiecţia este o operaţie unară care elimină anumite atribute ale unei relaţii producând o submulţime „pe verticală“ a acesteia. Suprimarea unor atribute poate avea ca efect apariţia unor tupluri duplicate, care trebuie eliminate.

Prin proiecţie se construieşte dintr-o relaţie *R*, o nouă relaţie:

a) ştergând din *R* atributele care nu sunt menţionate în parametrii proiecţiei;

b) eliminând dublurile care apar după ştergere.

Pentru a reprezenta operatorul proiecţie sunt utilizate diferite **notaţii**:

$\Pi_{A1, ..., Am}$ (R) PROJECT (R, A_1, ..., A_m)

$R[A_1, ..., A_m]$

unde A_1, A_2, ..., A_m sunt parametrii proiecţiei relativ la relaţia *R*.

***Exemplu*.** Să se obţină o listă ce conţine numele, prenumele şi sexul angajaţilor.

1. Proiecţie în algebra relaţională:

Rezultat = PROJECT(SALARIAT, nume, prenume, sex)

2. Proiecţie cu dubluri în SQL:

```
SELECT   nume, prenume, sex
FROM     salariat;
```

3. Proiecţie fără dubluri în SQL:

```
SELECT   DISTINCT nume, prenume, sex
FROM     salariat;
```

Operatorul SELECT

Selecţia (restrictia) este o operaţie unară care produce o submulţime pe „orizontală“ a unei relaţii *R.* Această submulţime se obţine prin extragerea tuplurilor din *R* care satisfac o condiţie specificată.

Sunt utilizate diferite notaţii:

$\sigma_{\text{condiţie}}(R)$ *R*[condiţie]

SELECT(*R*, condiţie) RESTRICT(*R*, condiţie).

***Exemplu*.** Să se obţină informaţii complete despre angajaţii de sex masculin.

1. Selecţie în algebra relaţională:

Rezultat = SELECT(SALARIAT, sex = ‘m’)

2. Selecţie în SQL:

```
SELECT   *
FROM     salariat
WHERE    sex = 'm';
```

Operatorul UNION

Reuniunea a două relaţii *R* şi *S* este mulţimea tuplurilor aparţinând fie lui *R*, fie lui *S*, fie ambelor relaţii.

Sunt utilizate notaţiile:

$R \cup S$

UNION(*R*, *S*)

OR(*R*, *S*)

APPEND(*R*, *S*).

***Exemplu*.** Să se obţină lista cu numele persoanelor fizice şi a subantreprenorilor.

```
SELECT   nume
FROM     subantreprenor
  UNION
SELECT   nume
FROM     pers_fizica;
```

Operatorul DIFFERENCE

Diferenţa a două relaţii *R* şi *S* este mulţimea tuplurilor care aparţin lui *R*, dar nu aparţin lui *S*. Diferenţa este o operaţie binară necomutativă care permite obţinerea tuplurilor ce apar numai într-o relaţie.

Sunt utilizate diferite notaţii:

R – *S*

DIFFERENCE(*R*, *S*)

REMOVE(*R*, *S*)

MINUS(*R*, *S*).

***Exemplu*.** Să se obţină lista cu numărul contractului, tipul contractului, valoarea investiţiei şi durata lucrării pentru contractele de subantrepriză pentru care valoarea investiţiei nu depăşeşte 60000$.

1. Diferenţă în algebra relaţională:

R=PROJECT(SELECT(CONTRACT, tip_contract=T),
nr_contract, tip_contract, val_investitie, durata_lucrare);
S=PROJECT(SELECT(CONTRACT, val_investitie > 60000),
nr_contract, tip_contract, val_investitie, durata_lucrare);
Rezultat = DIFFERENCE(*R*, *S*)

2. Diferenţa în SQL:

```
SELECT   nr_contract,tip_contract,
         val_investitie,durata_lucrare
FROM     contract
WHERE    tip_contract
   MINUS
SELECT   nr_contract,tip_contract,
         val_investitie,durata_lucrare
FROM     contract
WHERE    val_investitie>60000;
```

Evident diferenţa se poate referi la tabele diferite! Implementaţi cererea prin care se listează toate oraşele în care se află o filială, dar nici o proprietate.

Operatorul INTERSECT

Intersecţia a două relaţii *R* şi *S* este mulţimea tuplurilor care aparţin şi lui *R* şi lui *S*. Operatorul INTERSECT este un operator binar, comutativ, derivat:

$$R \cap S = R - (R - S)$$

$$R \cap S = S - (S - R).$$

Sunt utilizate diferite notaţii:

INTERSECT(*R*, *S*)

$R \cap S$

AND(*R*, *S*).

În anumite dialecte SQL există operator special (INTERSECT), care realizează această operaţie. Operatorii INTERSECT şi DIFFERENCE pot fi simulaţi în SQL (în cadrul comenzii SELECT) cu ajutorul opţiunilor EXISTS, NOT EXISTS, IN, != ANY.

Exemplu. Utilizând tabelele *agent_teritorial* şi *programator* să se obţină lista codurilor salariaţilor care sunt programatori, dar care lucrează şi ca agenţi teritoriali.

1. Intersecţie în algebra relaţională:

R = PROJECT(AGENT_TERITORIAL, cod_salariat);
S = PROJECT(PROGRAMATOR, cod_salariat),
Rezultat = INTERSECT(*R*, *S*).

2. Intersecţie în SQL:

```
SELECT   cod_salariat
FROM     agent_teritorial
  INTERSECT
SELECT   cod_salariat
FROM     programator;
```

3. Simularea intersecţiei în SQL:

```
SELECT   cod_salariat
FROM     programator p
WHERE EXISTS
     (SELECT  cod_salariat
      FROM    agent_teritorial a
      WHERE   p.cod_salariat=a.cod_salariat);
```

Operatorul PRODUCT

Fie *R* şi *S* relaţii de aritate *m*, respectiv *n*. Produsul cartezian al lui *R* cu *S* este mulţimea tuplurilor de aritate $m + n$ unde primele *m* componente formează un tuplu în *R*, iar ultimele *n* componente formează un tuplu în *S*.

Sunt utilizate diferite notaţii:

$R \times S$

PRODUCT(R, S)

TIMES(R, S).

Exemplu. Să se obţină lista tuturor posibilităţilor de investiţie în diverse obiective de către o firmă care este persoană juridică.

1. Produs cartezian în algebra relaţională:

R = PROJECT(PERS_JURIDICA, nume, cod_contractant);
S = PROJECT(OBIECTIV_INVESTITIE, denumire);
Rezultat = PRODUCT(R, S).

2. Produs cartezian în SQL:

```
SELECT   cod_contractant, nume, denumire
FROM     obiectiv_investitie, pers_juridica;
```

Operatorul DIVISION

Diviziunea este o operaţie binară care defineşte o relaţie ce conţine valorile atributelor dintr-o relaţie care apar **în toate** valorile atributelor din cealaltă relaţie.

Sunt utilizate diferite notaţii:

DIVIDE(R, S)

DIVISION(R, S)

$R \div S$.

Diviziunea conţine acele tupluri de dimensiune $n - m$ la care, adăugând orice tuplu din S, se obţine un tuplu din R.

Operatorul diviziune poate fi exprimat formal astfel:

$$R^{(n)} \div S^{(m)} = \{t^{(n-m)} \mid \forall\, s \in S, (t, s) \in R\} \text{ unde } n > m \text{ şi } S \neq \varnothing.$$

Operatorul DIVISION este legat de cuantificatorul universal ($\forall$) care nu există în SQL. Cuantificatorul universal poate fi însă simulat cu ajutorul cuantificatorului existenţial ($\exists$) utilizând relaţia:

$$\forall x\, P(x) \equiv \neg\, \exists\, x \neg\, P(x).$$

Prin urmare, operatorul DIVISION poate fi exprimat în SQL prin succesiunea a doi operatori NOT EXISTS.

Exemplu. Să se obţină codurile salariaţilor ataşaţi tuturor proiectelor pentru care s-a alocat un buget egal cu 1000.

1. Diviziune în algebra relaţională:

R = PROJECT(ATASAT_LA, cod_salariat, nr_proiect);
S = PROJECT(SELECT(PROIECT, buget = 1000), nr_proiect);
Rezultat = DIVISION(*R*, *S*).

2. Diviziune în SQL:

```
SELECT   UNIQUE cod_salariat
FROM     atasat_la sx
WHERE NOT EXISTS
      (SELECT *
      FROM    proiect pp
      WHERE   proiect.buget='1000'
      AND NOT EXISTS
         (SELECT  *
         FROM     atasat_la bb
         WHERE         pp.nr_proiect=bb.nr_proiect
         AND   bb.cod_salariat=sx.cod_salariat));
```

3. Simularea diviziunii cu ajutorul funcţiei COUNT:

```
SELECT   cod_salariat
FROM     atasat_la
WHERE    nr_proiect IN
     (SELECT  nr_proiect
      FROM    proiect
      WHERE   buget=1000)
GROUP BY cod_salariat
HAVING   COUNT(nr_proiect)=
              (SELECT  COUNT(*)
               FROM    proiect
               WHERE   buget=1000);
```

Operatorul JOIN

Operatorul de compunere (uniune) permite regăsirea informaţiei din mai multe relaţii corelate. Operatorul combină produsul cartezian, selecţia şi proiecţia.

Operatorul NATURAL JOIN

Operatorul de compunere naturală (NATURAL JOIN) combină tupluri din două relaţii *R* şi *S*, cu condiţia ca atributele comune să aibă valori identice.

Algoritmul care realizează compunerea naturală este următorul:

1. se calculează produsul cartezian $R \times S$;

2. pentru fiecare atribut comun A care defineşte o coloană în R şi o coloană în S, se selectează din $R \times S$ tuplurile ale căror valori coincid în coloanele $R.A$ şi $S.A$ (atributul $R.A$ reprezintă numele coloanei din $R \times S$ corespunzătoare coloanei A din R);
3. pentru fiecare astfel de atribut A se proiectează coloana $S.A$, iar coloana $R.A$ se va numi A.

Operatorul NATURAL JOIN poate fi exprimat formal astfel:

$$\text{JOIN}(R, S) = \Pi_{i1,\ldots,im}\, \sigma_{(R.A1 = S.A1) \wedge \ldots \wedge (R.Ak = S.Ak)}(R \times S),$$

unde A_1, ..., A_k sunt atributele comune lui R şi S, iar i_1, ..., i_m reprezintă lista componentelor din $R \times S$ (păstrând ordinea iniţială) din care au fost eliminate componentele $S.A_1$, ..., $S.A_k$.

Exemplu. Să se obţină informaţii complete despre angajaţi şi departamentele în care lucrează.

1. Operatorul de compunere naturală în algebra relaţională:

Rezultat = JOIN(SALARIAT, DEPARTAMENT).

2. Operatorul de compunere naturală în SQL:

```
SELECT   *
FROM     salariat, departament
WHERE    nr_depart = cod_departament;
```

Operatorul θ-JOIN

Operatorul **θ**-JOIN combină tupluri din două relaţii (nu neapărat corelate) cu condiţia ca valorile atributelor specificate să satisfacă o anumită condiţie specificată explicit în cadrul operaţiei.

Operatorul **θ**-JOIN este un operator derivat, fiind o combinaţie de produs scalar şi selecţie:

$$\text{JOIN}(R, S, \text{condiţie}) = \sigma_{\text{condiţie}}\,(R \times S)$$

Exemplu. Să se afişeze pentru fiecare salariat, codul acestuia şi grila sa de salarizare.

```
SELECT   empno, level
FROM     salgrade, emp
WHERE    sal BETWEEN losal AND hisal;
```

Exemplu. Să se obţină informaţii despre contractanţi (codul şi banca) şi obiectivele de investiţie asociate acestora (denumire, număr certificat de urbanizare) cu condiţia ca obiectivele să nu fie la aceeaşi adresă ca şi contractanţii.

1. Operatorul **θ**-JOIN în algebra relaţională:

R = PROJECT(CONTRACTANT, cod_contractant, banca);
S = PROJECT(OBIECTIV_INVESTITIE, denumire, nr_cert_urb);
Rezultat = JOIN(*R*, *S*, OBIECTIV_INVESTITIE.adresa <> CONTRACTANT.adresa).

2. Opratorul **θ**-JOIN în SQL:

```
SELECT   cod_contractant,banca, nr_cert_urb,
         denumire
FROM     contractant a,obiectiv_investitie b
WHERE    b.adresa <> a.adresa;
```

Operatorul SEMI-JOIN

Operatorul SEMI-JOIN conservă atributele unei singure relaţii participante la compunere şi este utilizat când nu sunt necesare toate atributele compunerii. Operatorul este asimetric.

Tupluri ale relaţiei R care participă în compunerea (naturală sau θ-JOIN) dintre relaţiile R şi S.

SEMI-JOIN este un operator derivat, fiind o combinaţie de proiecţie şi compunere naturală sau proiecţie şi θ-JOIN:

$$\text{SEMIJOIN}(R, S) = \Pi_M (\text{JOIN}(R, S))$$

$$\text{SEMIJOIN}(R, S, \text{condiţie}) = \Pi_M (\text{JOIN}(R, S, \text{condiţie})),$$

unde am notat prin *M* atributele relaţiei *R*.

Exemplu. Să se obţină informaţii referitoare la persoanele fizice (nume, buletin) care investesc în obiective cu caracter recreativ.

1. Operatorul SEMI-JOIN în algebra relaţională:

R = SELECT(OBIECTIV_INVESTITIE, denumire = ’cabana’ OR denumire = ’casa de vacanta’)
S = JOIN(PERS_FIZICA, *R*)
Rezultat = PROJECT(*S*, nume, buletin).

2. Operatorul SEMI-JOIN în SQL:

```
SELECT   nume,bi
FROM     pers_fizica a,obiectiv_investitie b
WHERE    a.cod_contractant = b.cod_contractant
AND      (denumire=’cabana’)OR (denumire= ’casa
          de vacanta’);
```

Operatorul OUTER JOIN

Operaţia de compunere externă combină tupluri din două relaţii pentru care sunt satisfăcute condiţiile de corelare. În cazul aplicării operatorului JOIN se pot pierde tupluri, atunci când există un tuplu în una din relaţii pentru care nu există nici un tuplu în cealaltă relaţie, astfel încât să fie satisfăcută relaţia de corelare.

Operatorul elimină acest inconvenient prin atribuirea valorii *null* valorilor atributelor care există într-un tuplu din una dintre relaţiile de intrare, dar care nu există şi în cea de-a doua relaţie.

Practic, se realizează compunerea a două relaţii *R* şi *S* la care se adaugă tupluri din *R* şi *S*, care nu sunt conţinute în compunere, completate cu valori *null* pentru atributele care lipsesc.

Compunerea externă poate fi: LEFT, RIGHT, FULL. De exemplu, OUTER JOIN LEFT reprezintă compunerea în care tuplurile din R, care nu au valori similare în coloanele comune cu relaţia S, sunt de asemenea incluse în relaţia rezultat.

Exemplu. Să se obţină informaţii referitoare la persoanele fizice care sunt investitori (chiar dacă nu au investit în obiective industriale) şi la obiectivele de investiţie industriale (chiar şi cele care nu sunt construite de persoane fizice).

R = SELECT(OBIECTIV_INVESTITIE, denumire = 'industrial')
Rezultat = OUTERJOIN(PERS_FIZICA, *R*).

Operatorii algebrei relaţionale pot fi reprezentaţi grafic cu ajutorul unor simboluri speciale. → curs!

Operaţii adiţionale: complement, despicare, închidere tranzitivă. Funcţii asociate: MIN, MAX, COUNT, AVG, SUM, VAR, STD etc.

Evaluarea şi optimizarea interogărilor

Procesarea interogărilor

O **expresie** a algebrei relaţionale este constituită din relaţii legate prin operaţii din algebra relaţională. O expresie se poate reprezenta grafic cu ajutorul unui arbore, numit **arbore algebric**, în care nodurile corespund operatorilor din cadrul expresiei respective.

Evaluarea unei expresii presupune efectuarea prelucrărilor indicate de operatorii din expresie în ordinea apariţiilor sau în ordinea fixată prin paranteze. Rezultatul evaluării unei expresii este o relaţie derivată din relaţiile menţionate ca operanzi în cadrul expresiei.

Două expresii sunt **echivalente**, dacă în urma evaluării lor se obţine ca rezultat aceeaşi relaţie.

Exemple referitoare la moduri echivalente de exprimare a unei cereri (vor fi rezolvate la curs!).

1. Informaţii despre salariaţii care nu contribuie la machetarea nici unei publicaţii, dar au retribuţia mai mare decât o valoare dată.

2. Codul şi numele subantreprenorilor care au realizat lucrări specializate la obiective case de vacanţă sau cabane.

3. Codurile şi telefoanele investitorilor, valoarea si durata de execuţie a investitiilor a caror valoare este între două limite specificate.

4. Perioada de desfăşurare şi preţul ofertelor care încep după 1 ianuarie 2003 şi sunt:

- sejururi la munte;
- excursii în care autocarele sunt conduse de şoferi angajaţi după 1 mai 1987 şi supravegheate de ghizi ce cunosc limba engleză care au făcut specializare în Suedia.

În majoritatea sistemelor de gestiune, şi în special în cele relaţionale, interfaţa cu utilizatorul este de tip **neprocedural**. Utilizatorul defineşte datele pe care doreşte să le vizualizeze fără a da algoritmii de acces. Sistemul trebuie să convertească cererea utilizatorului:

- într-o cerere optimală;
- în proceduri de acces optimal la datele fizice.

Garantarea absolută a performanţelor optime pentru procesorul limbajului relaţional este imposibilă. Corectă ar fi utilizarea cuvântului „ameliorare“ în locul cuvântului „optimizare“.

Evaluarea unei interogări se efectuează în trei etape.

- ❒ Analiza cererii presupune studierea sintactică şi semantică a cererii pentru a verifica corectitudinea sa şi a simplifica criteriul de căutare.
- ❒ Ordonanţarea presupune descompunerea cererii într-o mulţime de operaţii elementare şi determinarea unei ordini optimale a acestor operaţii. Operaţiile sunt, în general, cele ale algebrei relaţionale. La sfârşitul etapei se obţine un plan de execuţie pentru cerere.
- ❒ Execuţia presupune efectuarea (paralel şi/sau secvenţială) operaţiilor elementare furnizate de planul de execuţie pentru a obţine rezultatul cererii.

Presupunem că utilizatorul transmite sistemului de gestiune o cerere exprimată prin ordine SQL. Pentru a răspunde cererii, SGBD-ul trebuie să înţeleagă cererea utilizatorului. Cererea trebuie să fie corectă sintactic, datele trebuie să fie disponibile utilizatorului şi trebuie localizate analizând diferite drumuri de acces la ele. Aceste funcţii sunt realizate de SGBD cu ajutorul a două module funcţionale care comunică permanent:

- **analizorul cererilor**, care asigură verificarea sintactică şi semantică a cererii, localizarea datelor implicate în cerere (găsirea adresei blocurilor ce conţin datele), furnizarea planului de execuţie.
- **administratorul datelor** (executorul), care execută efectiv cererea (primeşte planurile de execuţie furnizate de modulul de optimizare şi le execută). Execuţia presupune căutarea blocurilor ce conţin datele şi transferul blocurilor în memoria *cache.*

Ideea generală:

cerere→ arbore algebric (nu este unic) → plan de executie → optimizare

Un plan de execuţie implică o secvenţă de paşi pentru evaluarea cererii (în mod obişnuit, fiecare pas din planul de execuţie corespunde unei operaţii relaţionale) precum şi metoda care va fi folosită pentru evaluarea operaţiei. De obicei, pentru o operaţie relaţională dată, există mai multe metode ce pot fi folosite pentru evaluarea acesteia.

Două planuri de execuţie diferite care au întotdeauna acelaşi rezultat se numesc echivalente. **Planuri de execuţie echivalente pot avea diferite costuri**. Scopul optimizării cererilor este de a găsi, printre diversele planuri de execuţie echivalente, pe acela de cost minim. Într-un sistem centralizat, costul evaluării unei cereri este suma a două componente, costul I/O (transferuri de date) şi costul CPU (verificare de condiţii, operaţii *join* etc.).

Ordinea de execuţie a operaţiilor

O interogare constă dintr-un număr de operaţii. Ordinea în care se efectuează operaţiile are un rol important în evaluarea costului necesar realizării interogării.

Există două modalităţi de abordare pentru a determina ordinea de execuţie a operaţiilor:

- algebric;
- bazat pe estimarea costului.

Ambele folosesc o mulţime de reguli care permit transformarea unui plan de execuţie (reprezentat ca o expresie scrisă în termenii algebrei relaţionale) în altul, echivalent.

Optimizarea cererilor bazată pe algebra relaţională se realizează în două etape:

- se exprimă cererile sub forma unor expresii algebrice relaţionale;
- se aplică acestor expresii transformări algebrice care conduc la expresii echivalente, dar care vor fi executate mai eficient.

Procesul de transformare a cererilor se realizează conform unei strategii de optimizare care poate să fie:

- independentă de modul de memorare a datelor (strategie generală);
- dependentă de modul de memorare (strategie specifică unui anumit SGBD).

Proprietăţile operatorilor algebrei relaţionale

Proprietatea 1. Comutativitatea operaţiilor de *join* şi produs cartezian:

$$\text{JOIN}(R1, R2) = \text{JOIN}(R2, R1)$$
$$R1 \times R2 = R2 \times R1$$

Proprietatea 2. Asociativitatea operaţiilor de *join* şi produs cartezian:

$$\text{JOIN}(\text{JOIN}(R1, R2), R3) = \text{JOIN}(R1, \text{JOIN}(R2, R3))$$
$$(R1 \times R2) \times R3 = R1 \times (R2 \times R3)$$

Proprietatea 3. Compunerea proiecţiilor:

$$\Pi_{A1,\ldots,Am}(\Pi_{B1,\ldots,Bn}(R)) = \Pi_{A1,\ldots,Am}(R),$$

unde $\{A_1, A_2,\ldots,A_m\} \subseteq \{B_1, B_2,\ldots,B_n\}$.

Proprietatea 4. Compunerea selecţiilor:

$$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond1 \wedge cond2}(R) = \sigma_{cond2}(\sigma_{cond1}(R)),$$

unde am notat prin *cond* condiţia după care se face selecţia.

Proprietatea 5. Comutarea selecţiei cu proiecţia:

$$\Pi_{A1,...,Am}(\sigma_{cond}(R)) = \sigma_{cond}(\Pi_{A1,...,Am}(R)),$$

unde condiţia după care se face selecţia implică numai atributele $A_1,...,A_m$.

Dacă condiţia implică şi atributele $B_1,...,B_n$, care nu aparţin mulţimii $\{A_1,...,A_m\}$, atunci:

$$\Pi_{A1,...,Am}(\sigma_{cond}(R)) = \Pi_{A1,...,Am}(\sigma_{cond}(\Pi_{A1,...,Am,B1,...,Bn}(R)))$$

Proprietatea 6. Comutarea selecţiei cu produsul cartezian:

Dacă toate atributele menţionate în condiţia după care se face selecţia sunt atribute ale relaţiei $R1$, atunci:

$$\sigma_{cond}(R1 \times R2) = \sigma_{cond}(R1) \times R2$$

Dacă condiţia este de forma $cond1 \wedge cond2$ şi dacă $cond1$ implică numai atribute din $R1$, iar $cond2$ implică numai atribute din $R2$, atunci

$$\sigma_{cond}(R1 \times R2) = \sigma_{cond1}(R1) \times \sigma_{cond2}(R2)$$

Dacă $cond1$ implică numai atribute din $R1$, iar $cond2$ implică atribute atât din $R1$ cât şi din $R2$, atunci:

$$\sigma_{cond}(R1 \times R2) = \sigma_{cond2}(\sigma_{cond1}(R1) \times R2)$$

Proprietatea 7. Comutarea selecţiei cu reuniunea:

$$\sigma_{cond}(R1 \cup R2) = \sigma_{cond}(\mathrm{R1}) \cup \sigma_{cond}(R2)$$

Proprietatea 8. Comutarea selecţiei cu diferenţa:

$$\sigma_{cond}(R1 - R2) = \sigma_{cond}(R1) - \sigma_{cond}(R2)$$

Proprietatea 9. Comutarea proiecţiei cu reuniunea:

$$\Pi_{A1,...,Am}(R1 \cup R2) = \Pi_{A1,...,Am}(R1) \cup \Pi_{A1,...,Am}(R2)$$

Proprietatea 10. Comutarea proiecţiei cu produsul cartezian:

Dacă $A_1,...,A_m$ este o listă de atribute ce apar în schemele relaţionale $R1$ şi $R2$ şi dacă lista este formată din atribute aparţinând lui $R1$ (notate prin $B_1,...,B_n$) şi din atribute aparţinând lui $R2$ (notate prin $C_1,...,C_k$) atunci:

$$\Pi_{A1,...,Am}(R1 \times R2) = \Pi_{B1,...,Bn}(R1) \times \Pi_{C1,...,Ck}(R2)$$

Proprietatea 11. Compunerea proiecţiei cu operaţia *join*:

Dacă $A_1,...,A_m$ este o listă de atribute ce apar în schemele relaţionale $R1$ şi $R2$ şi dacă lista este formată din atribute aparţinând lui $R1$ (notate prin $B_1,...,B_n$) şi din atribute aparţinând lui $R2$ (notate prin $C_1,...,C_k$) atunci:

$$\Pi_{A1,...,Am}(\mathrm{JOIN}(R1,R2,D)) = \Pi_{A1,...,Am}(\mathrm{JOIN}(\Pi_{D,B1,...,Bn}(R1), \Pi_{D,C1,...,Ck}(R2),D),$$

unde am notat prin JOIN($R1$, $R2$, D) operaţia de compunere naturală între $R1$ şi $R2$ după atributul comun D.

Proprietatea 12. Compunerea selecţiei cu operaţia *join*:

$$\sigma_{cond}\,(\text{JOIN}\,(R1, R2, D)) = \sigma_{cond}\,(\text{JOIN}\,(\Pi_{D,A}\,(R1), \Pi_{D,A}\,(R2), D)),$$

unde *A* reprezintă atributele care apar în condiţia după care se face selecţia.

Reguli de optimizare frecvent folosite:

Regula de optimizare 1. Selecţiile se execută cât mai devreme posibil. Motivaţia acestei reguli este că selecţiile reduc substanţial dimensiunea relaţiilor. Regula de transformare 4 poate fi folosită pentru a separa două sau mai multe selecţii în selecţii individuale care pot fi distribuite *join*-ului sau produsului cartezian folosind comutarea selecţiei cu *join*-ul.

Regula de optimizare 2. Produsurile carteziene se înlocuiesc cu *join*-uri, ori de câte ori este posibil. Un produs cartezian între două relaţii este de obicei mult mai scump (ca şi cost) decât un *join* între cele două relaţii, deoarece primul generează concatenarea tuplurilor în mod exhaustiv şi poate genera un rezultat foarte mare. Această transformare se poate realiza folosind legătura dintre produs cartezian, *join* şi selecţie.

Regula de optimizare 3. Dacă sunt mai multe *join*-uri atunci cel care se execută primul este cel mai restrictiv. Un *join* este mai restrictiv decât altul dacă produce o relaţie mai mică. Se poate determina care *join* este mai restrictiv pe baza factorului de selectivitate sau cu ajutorul informaţiilor statistice. Algebric, acest lucru se poate realiza folosind regula de transformare 2.

Regula de optimizare 4. Proiecţiile se execută la început pentru a îndepărta atributele nefolositoare. Dacă un atribut al unei relaţii nu este folosit în operaţiile ulterioare atunci trebuie îndepărtat. În felul acesta se va folosi o relaţie mai mică în operaţiile ulterioare. Aceasta se poate realiza folosind comutarea proiecţiei cu *join*-ul.

Exemple curs!!!

Regulile lui Codd

Caracteristici ale modelului relaţional:

- nu există tupluri identice;
- ordinea liniilor şi a coloanelor este arbitrară;
- articolele unui domeniu sunt omogene;
- fiecare coloană defineşte un domeniu distinct şi nu se poate repeta în cadrul aceleiaşi relaţii;

- toate valorile unui domeniu corespunzătoare tuturor cazurilor nu mai pot fi descompuse în alte valori (sunt atomice).

Avantajele modelului relaţional:

- fundamentare matematică riguroasă;
- independenţă fizică a datelor;
- posibilitatea filtrărilor;
- existenţa unor structuri de date simple;
- realizarea unei redundanţe minime;
- supleţe în comunicarea cu utilizatorul neinformatician.

Ca **limite** ale modelului relaţional putem menţiona:

- rămâne totuşi redundanţă,
- ocupă spaţiu,
- apar fenomene de inconsistenţă,
- nu există mecanisme pentru tratarea optimă a cererilor recursive,
- nu lucrează cu obiecte complexe,
- nu există mijloace perfecţionate pentru exprimarea constrângerilor de integritate,
- nu realizează gestiunea totala a datelor distribuite,
- nu realizează gestiunea cunoştinţelor.

În anul 1985, E.F. Codd a publicat un set de 13 reguli în raport cu care un sistem de gestiune a bazelor de date poate fi apreciat ca relaţional. Nici un sistem de gestiune a bazelor de date pus în vânzare pe piaţa comercială nu respectă absolut toate regulile definite de Codd, dar acest lucru nu împiedică etichetarea acestor sisteme drept relaţionale.

Nu trebuie apreciat un SGBD ca fiind relaţional sau nu, ci măsura în care acesta este relaţional, deci numărul regulilor lui Codd pe care le respectă.

Regula 1 – regula gestionării datelor. *Un SGBD relaţional trebuie să fie capabil să gestioneze o bază de date numai prin posibilităţile sale relaţionale.*

Regula 2 – regula reprezentării informaţiei. *Într-o bază de date relaţională, informaţia este reprezentată la nivel logic sub forma unor tabele ce poartă numele de relaţii*.

Regula 3 – regula accesului garantat la date. *Fiecare valoare dintr-o bază de date relaţională trebuie să poată fi adresată în mod logic printr-o combinaţie formată din numele relaţiei, valoarea cheii primare şi numele atributului.*

Regula 4 – regula reprezentării informaţiei necunoscute. *Un sistem relaţional trebuie să permită utilizatorului definirea unui tip de date numit „null“ pentru reprezentarea unei informaţii necunoscute la momentul respectiv.*

Regula 5 – regula dicţionarelor de date. *Asupra descrierii bazelor de date* (*informaţii relative la relaţii, vizualizări, indecşi etc.*) *trebuie să se poată aplica aceleaşi operaţii ca şi asupra datelor din baza de date.*

Regula 6 – regula limbajului de interogare. *Trebuie să existe cel puţin un limbaj pentru prelucrarea bazei de date*.

Regula 7 – regula de actualizare a vizualizării. *Un SGBD trebuie să poată determina dacă o vizualizare poate fi actualizată şi să stocheze rezultatul interogării într-un dicţionar de tipul unui catalog de sistem.*

Regula 8 – regula limbajului de nivel înalt. *Regulile de prelucrare asupra unei relaţii luată ca întreg sunt valabile atât pentru operaţiile de regăsire a datelor, cât şi asupra operaţiilor de inserare, actualizare şi ştergere a datelor*.

Regula 9 – regula independenţei fizice a datelor: *Programele de aplicaţie şi activităţile utilizatorilor nu depind de modul de depunere a datelor sau de modul de acces la date.*

Regula 10 – regula independenţei logice a datelor. *Programele de aplicaţie trebuie să fie transparente la modificările de orice tip efectuate asupra datelor*.

Regula 11 – regula independenţei datelor din punct de vedere al integrităţii. *Regulile de integritate trebuie să fie definite într-un sublimbaj relaţional, nu în programul de aplicaţie*.

Regula 12 – regula independenţei datelor din punct de vedere al distribuirii. *Distribuirea datelor pe mai multe calculatoare dintr-o reţea de comunicaţii de date, nu trebuie să afecteze programele de aplicaţie*.

Regula 13 – regula versiunii procedurale a unui SGBD. *Orice componentă procedurală a unui SGBD trebuie să respecte aceleaşi restricţii de integritate ca şi componenta relaţională.*

Deoarece regulile lui Codd sunt prea severe pentru a fi respectate de un SGBD operaţional, s-au formulat criterii minimale de definire a unui sistem de gestiune relaţional.

Un SGBD este **minimal relaţional** dacă:

- toate datele din cadrul bazei sunt reprezentate prin valori în tabele;
- nu există pointeri observabili de către utilizator;
- sistemul suportă operatorii relaţionali de proiecţie, selecţie şi compunere naturală, fără limitări impuse din considerente interne.

Un SGBD este **complet relaţional** dacă este minimal relaţional şi satisface în plus condiţiile:

- sistemul suportă restricţiile de integritate de bază (unicitatea cheii primare, constrângerile referenţiale, integritatea entităţii).
- sistemul suportă toate operaţiile de bază ale algebrei relaţionale.

NORMALIZAREA RELAŢIILOR

În procesul modelării unei baze de date relaţionale, o etapă importantă o reprezintă **normalizarea relaţiilor conceptuale** (Codd, 1972), adică obţinerea de relaţii „moleculare“fără a pierde nimic din informaţie pentru a elimina:

- redundanţa;
- anomaliile reactualizării informaţiilor.

Tehnica normalizării permite obţinerea unei scheme conceptuale rafinate printr-un proces de ameliorare progresivă a unei scheme conceptuale iniţiale a bazei de date relaţionale. După fiecare etapă de ameliorare, relaţiile bazei de date ating un anumit grad de perfecţiune, deci se află într-o anumită formă normală. Trecerea unei relaţii dintr-o formă normală în alta, presupune eliminarea unei anumit tip de dependenţe nedorite, care sunt transformate în dependenţe admisibile, adică dependenţe care nu provoacă anomalii.

Procesul de ameliorare a schemei conceptuale **trebuie**:

- să garanteze **conservarea datelor**, adică în schema conceptuală finală trebuie să figureze toate datele din cadrul schemei iniţiale;
- să garanteze **conservarea dependenţelor** dintre date, adică în schema finală fiecare dependenţă trebuie să aibă determinantul şi determinatul în schema aceleiaşi relaţii;
- să reprezinte o **descompunere minimală** a relaţiilor iniţiale, adică nici una din relaţiile care compun schema finală nu trebuie să fie conţinută într-o altă relaţie din această schemă.

Există două metode pentru a modela baze de date relaţionale fără anomalii sau pierderi de informaţie.

- **Schema descompunerii** pleacă de la o schemă relaţională universală ce conţine toate atributele BD. Schema se descompune prin proiecţii succesive în subrelaţii. Descompunerea se opreşte când continuarea ei ar duce la pierderi de informaţie. Algoritmii de descompunere se bazează, în general, pe descrierea formală a dependenţei dintre atribute.
- **Schema sintezei** pleacă de la o mulţime de atribute independente. Utilizând proprietăţi de semantică şi legături între atribute se pot compune noi relaţii, astfel încât, acestea să nu sufere de anumite anomalii. Algoritmii se bazează, în general, pe teoria grafurilor pentru a reprezenta legătura între atribute.

Dependenţe funcţionale

O relaţie universală este o relaţie ce grupează toate atributele care modelează sistemul real cercetat. Fie *E*, mulţimea dependenţelor considerate de proiectantul bazei pentru o schemă relaţională sau pentru o relaţie universală. Plecând de la o mulţime de proprietăţi formale ale dependenţelor, proprietăţi considerate drept reguli de deducţie (axiome), poate fi obţinută mulţimea maximală de dependenţe asociate lui *E*. Această mulţime defineşte **închiderea** lui *E*.

Fie *E* mulţimea dependenţelor unei relaţii şi p_1, p_2, ..., p_r, $r \geq 1$, proprietăţi formale ale acestor dependenţe. Dacă există o mulţime E', astfel încât orice dependenţă a mulţimii *E* este derivabilă din E' prin aplicarea proprietăţilor p_1, p_2, ..., p_r, atunci mulţimea E' defineşte **acoperirea** lui *E* pentru proprietăţile p_1, p_2, ..., p_r.

E' este o **acoperire minimală** pentru *E*, dacă nu există nici o submulţime proprie, nevidă a lui E' care să fie o acoperire pentru *E*. **Evident, *E* şi E' au închideri identice, deci dispun de acelaşi potenţial informaţional!**

Fie $R(A_1, A_2, ..., A_n)$ o schemă relaţională şi fie *X*, *Y* submulţimi de atribute ale lui *R*. *X* **determină funcţional** *Y* sau *Y* depinde funcţional (FD) de *X*, dacă pentru orice relaţie *r* (valoare curentă a lui *R*) nu există două tupluri care să aibă aceleaşi valori pentru atributele lui *X* şi să aibă valori diferite pentru cel puţin un atribut din *Y*. Cu alte cuvinte, o valoare a lui *X*, determină unic o valoare a lui *Y*.

Notaţia utilizată pentru desemnarea dependenţei funcţionale este $\overline{X \rightarrow Y}$. X este numit **determinant,** iar Y este numit **determinat** (sau dependent). Dependenţa funcţională $X \rightarrow Y$ este **trivială** dacă $Y \subseteq X$.

Comparând toate submulţimile de atribute ale unei relaţii şi determinând legăturile dintre ele, se pot obţine toate dependenţele funcţionale pe care o relaţie le satisface. Această abordare **nu** este eficientă, consumând mult timp.

Există posibilitatea ca, ştiind anumite dependenţe funcţionale şi utilizând reguli de deducţie, să fie obţinute **toate** dependenţele funcţionale.

Fie X, Y, Z, W mulţimi de atribute ale unei scheme relaţionale R şi fie următoarele axiome:

Ax1 – reflexivitate. $X \rightarrow X$. Mai general, dacă $Y \subseteq X$, atunci $X \rightarrow Y$.

Ax2 – creşterea determinantului. Pot fi considerate următoarele formulări echivalente pentru această axiomă.

1. Dacă $X \rightarrow Y$ şi $X \subseteq Z$, atunci $Z \rightarrow Y$.
2. Dacă $X \rightarrow Y$ şi $W \subseteq Z$, atunci $X \cup Z \rightarrow Y \cup W$.
3. Dacă $X \rightarrow Y$ atunci $X \cup Z \rightarrow Y \cup Z$.
4. Dacă $X \rightarrow Y$ atunci $X \cup Z \rightarrow Y$.

Ax3 – tranzitivitate. Dacă $X \rightarrow Y$ şi $Y \rightarrow Z$, atunci $X \rightarrow Z$.

O mulţime de axiome este **completă** dacă şi numai dacă plecând de la o mulţime de dependenţe E se pot obţine toate dependenţele închiderii lui E, utilizând axiomele mulţimii.

O mulţime de axiome este **închisă** dacă şi numai dacă plecând de la o mulţime de dependenţe E, nu poate fi dedusă cu ajutorul axiomelor o dependenţă care nu aparţine închiderii lui E. (nu obţin altele!)

Ullman a demonstrat că axiomele Ax1 – Ax3, numite axiomele lui Amstrong, reprezintă o mulţime închisă şi completă de axiome. Consecinţa acestui rezultat este că **închiderea lui *E* reprezintă mulţimea dependenţelor deduse din *E*, prin aplicarea axiomelor lui Amstrong!!!**

Nu toate dependenţele funcţionale sunt folositoare pentru modelarea relaţională. O dependenţă funcţională $X \rightarrow Y$ se numeşte **dependenţă funcţională totală** (FT), dacă şi numai dacă nu există nici o submulţime proprie $X' \subset X$, astfel încât $X' \rightarrow Y$. Dacă există o submulţime proprie $X' \subset X$, astfel încât $X' \rightarrow Y$, atunci dependenţa funcţională $X \rightarrow Y$ este **parţială**. În axioma Ax2, dependenţa $Z \rightarrow Y$ este o dependenţă funcţională parţială.

În cazul dependenţei funcţionale totale, axiomele lui Amstrong se reduc la o axiomă unică şi anume pseudo-tranzitivitatea:

dacă $X \rightarrow Y$ şi $W \cup Y \rightarrow Z$, atunci $W \cup X \rightarrow Z$.

Această axiomă este o regulă de deducţie completă pentru total dependenţe:

- pseudo-tranzitivitatea implică tranzitivitatea ($W = \varnothing$);
- reflexivitatea nu poate fi utilizată pentru a obţine dependenţe totale;
- reflexivitatea şi pseudo-tranzitivitatea implică creşterea.

Dacă F este o mulţime de dependenţe funcţionale totale, atunci **închiderea pseudo-tranzitivă** F^+ a acestei mulţimi este reuniunea mulţimilor dependenţelor funcţionale totale care pot fi obţinute din F folosind axioma de pseudo-tranzitivitate.

Două mulţimi de dependenţe funcţionale totale sunt **echivalente** dacă au închideri pseudo-tranzitive identice. Pentru a modela scheme relaţionale se consideră mulţimi minimale de dependenţe funcţionale totale, capabile să genereze toate închiderile pseudo-tranzitive. Aceste mulţimi definesc acoperiri minimale.

O mulţime de dependenţe funcţionale totale F^* asociată unei mulţimi de atribute A defineşte o **acoperire minimală** dacă satisface următoarele proprietăţi:

- nici o dependenţă funcţională din F^* nu este redundantă;
- toate dependenţele funcţionale totale între submulţimi ale lui A sunt în închiderea pseudo-tranzitivă a lui F^*.

Orice mulţime de dependenţe totale are cel puţin o acoperire minimală. Alegerea acoperirii minimale este punctul de start în modelarea schemelor relaţionale.

Dependenţele funcţionale între atributele bazei pot fi reprezentate grafic. Fie $A = \{A_1, A_2, ..., A_n\}$ o mulţime de atribute şi fie o mulţime de dependenţe funcţionale $\{X_i \rightarrow A_j\}$, unde X_i este o submulţime a lui A.

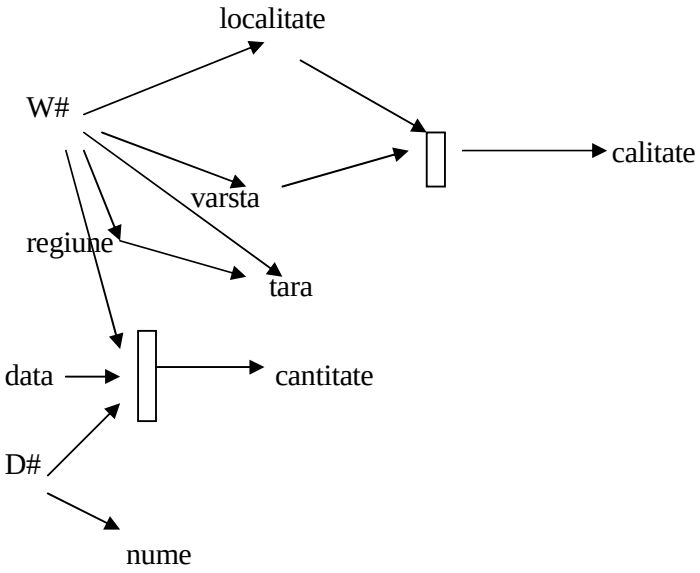
Graful dependenţelor funcţionale este un graf direcţionat bipartit, definit astfel:

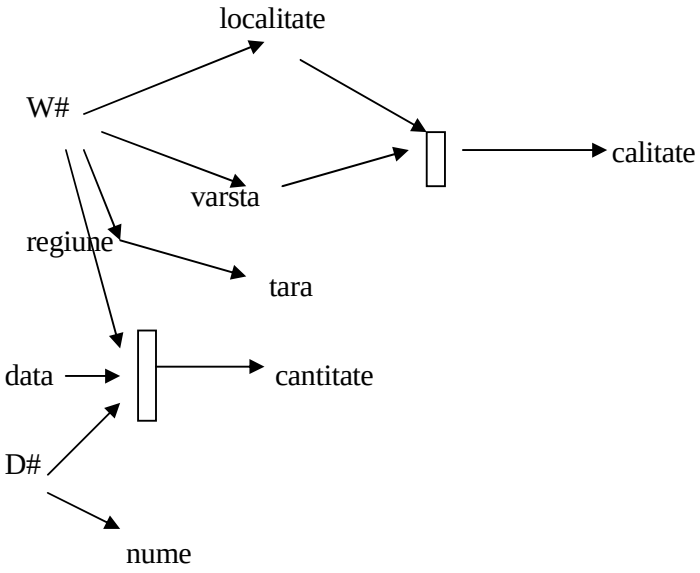
1. pentru fiecare atribut A_j există un singur nod având eticheta A_j;
2. pentru fiecare dependenţă funcţională de forma $A_i \rightarrow A_j$, există un arc de la A_i la A_j;

3. pentru fiecare dependenţă funcţională de forma $X_i \rightarrow A_j$, unde mulţimea X_i este definită de $X_i = \{Ai_1, ..., Ai_p\}$ cu $p > 1$, există un nod auxiliar etichetat prin X_i şi o mulţime de arce plecând de la $Ai_1, ..., Ai_p$ pentru a obţine pe X_i şi printr-un arc adiţional de la X_i la A_j. Nodurile X_i se reprezintă prin dreptunghiuri.

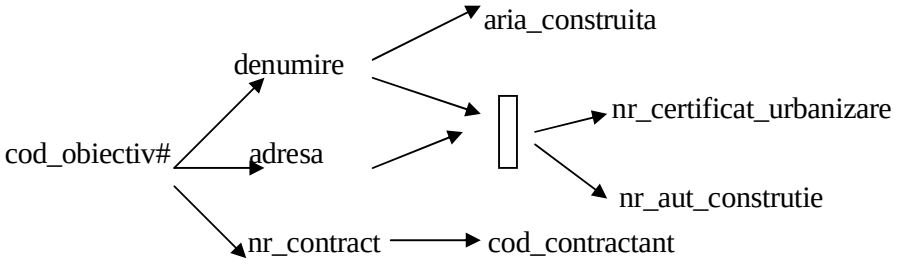
Exemplu:

1. Graful dependenţelor funcţionale pentru schema relaţională CONSUMATOR_DE_VIN(W#, localitate, varsta, calitate, regiune, tara, D#, nume, data, cantitate) şi acoperirea minimală.





2. Graful dependenţelor funcţionale pentru schema relaţională OBIECTIV_INVESTITIE. Dependentele sunt deduse din regulile impuse de beneficiar!



Necesitatea normalizării

Anomaliile care apar în lucrul cu baza de date se produc datorită dependenţelor care există între datele din cadrul relaţiilor bazei de date. Dependenţele sunt plasate greşit în tabele!!!

Avion

A#	nume	capacitate	localitate
1	AIRBUS	250	PARIS
2	AIRBUS	250	PARIS
3	AIRBUS	250	LONDRA
4	CAR	100	PARIS
5	B707	150	LONDRA
6	B707	150	LONDRA

Constrângere:
toate avioanele cu acelaşi nume au aceeaşi capacitate.

Datorită dependenţei introduse pot exista: anomalii la inserare, modificare sau ştergere, redundanţă în date, probleme de reconexiune.

1. **Redundanţă logică**. Cuplul (AIRBUS, 250) apare de trei ori.
2. **Anomalie la inserţie**. S-a cumpărat un B727 cu 150 locuri. El poate fi inserat în relaţia AVION doar dacă se defineşte o nouă valoare pentru cheia primară.
3. **Anomalie la ştergere**. Dacă este ştearsă înregistrarea pentru care A# este 4, atunci se pierde informaţia că un avion CAR are capacitatea 100.

4. **Anomalie la modificare**. Dacă se modifică capacitatea lui B$\overline{707}$ de la 150 la 170, atunci costul modificării este mare pentru a modifica toate înregistrările, iar dacă se modifică doar o înregistrare atunci constrângerea nu va mai fi verificată.
5. **Problema reconexiunii**. Considerăm schemele relaţionale:
   AVION1 = PROJECT(AVION, A#, nume)
   AVION22 = PROJECT(AVION, nume, capacitate, localitate)
   AVION3 = JOIN(AVION1, AVION2).
   Se observă că schema AVION3 este diferită de AVION.
   Apare un tuplu nou: (3, AIRBUS, 250, PARIS).

Anomaliile au apărut datorită dependenţei funcţionale (constrângerii) introduse anterior!!!

Normalizarea are drept scop:

- suprimarea redundanţei logice,
- evitarea anomaliilor la reactualizare,
- rezolvarea problemei reconexiunii.

Există o teorie matematică a normalizării al cărei autor este E.F. Codd. Soluţia: construirea unor tabele standard (forme normale).

Normalizarea este procesul reversibil de transformare a unei relaţii, în relaţii de structură mai simplă. Procesul este reversibil în sensul că nici o informaţie nu este pierdută în timpul transformării. O relaţie este într-o formă normală particulară dacă ea satisface o mulţime specificată de constrângeri.

Procesul normalizării se realizează plecând de la o relaţie universală ce conţine toate atributele sistemului de modelat, plus o mulţime de anomalii. Orice formă normală se obţine aplicând o schemă de descompunere. Există două tipuri de descompuneri.

- **Descompuneri ce conservă dependenţele**. Această descompunere presupune desfacerea relaţiei R în proiecţiile R_1, R_2, ..., R_k, astfel încât dependenţele lui R sunt echivalente (au închideri pseudo-tranzitive identice) cu reuniunea dependenţelor lui R_1, R_2, ..., R_k.
- **Descompuneri fără pierderi de informaţie** (*L-join*). Această descompunere presupune desfacerea relaţiei R într-o mulţime de proiecţii R_1, R_2, ..., R_j, astfel încât pentru orice realizare a lui R este adevărată relaţia:

$$R = \text{JOIN}(\Pi_{B1}(R), \Pi_{B2}(R), ..., \Pi_{Bj}(R))$$

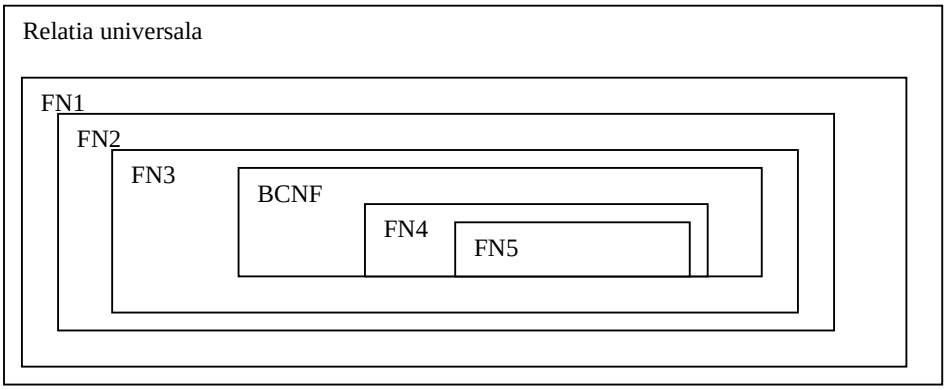
unde, pentru $1 \le k \le j$, B_k reprezintă mulţimea atributelor corespunzătoare proiecţiei R_k ($R_k = \Pi_{Bk}(R)$). Prin urmare, relaţia iniţială poate fi reconstruită considerând compunerea naturală a relaţiilor obţinute prin descompunere. Formele normale sunt obţinute prin descompuneri fără pierderi de informaţie.

O descompunere fără pierdere de informaţie, utilizată în procesul normalizării, este dată de **regula Casey-Delobel**:

Fie $R(A)$ o schemă relaţională şi fie α, β, γ o partiţie a lui A. Presupunem că α determină funcţional pe β. Atunci:

$$R(A) = \text{JOIN}(\Pi_{\alpha\cup\beta}(R), \Pi_{\alpha\cup\gamma}(R)).$$

$\alpha\cup\beta \rightarrow$ mulţimea atributelor care intervin în dependenţele funcţionale;
$\alpha\cup\gamma \rightarrow$ reprezintă reuniunea determinantului cu restul atributelor lui A.



Pentru exemplul analizat anterior:
α = {nume},
β = {capacitate},
γ = {A#, localitate}.

Aplicând Casey-Delobel se obţin schemele:

AVION1(nume#, capacitate)
AVION2)A#, nume, localitate).

Se observă că anomaliile comentate au dispărut!

AVION1

Nume	Capacitate
AIRBUS	150
CAR	100
B707	150

AVION2

A#	Nume	Localitate
1	AIRBUS	PARIS
2	AIRBUS	PARIS
3	AIRBUS	LONDRA
4	CAR	PARIS
5	B707	LONDRA
6	B707	LONDRA

Forma normală (FN1)

O relaţie este în prima formă normală dacă fiecărui atribut care o compune îi corespunde o valoare indivizibilă (atomică).

Exemplu: variante pentru a implementa FN1 pentru tabelul MASINA:

Persoana	Vehicul
Eu	R25 - W14 - R21
Tu	205
El	R5 - 305
noi	BX - 305 - R12 - R25

Varianta 1

Persoana	Vehicul
Eu	R25
Eu	W14
Eu	R21
Tu	205
El	R5
El	305
Noi	BX
Noi	305
Noi	R12
Noi	R25

Varianta 2

Persoana	Prima	Doi	Trei	Patru
Eu	R25	W14	R21	
Tu	205			
El	R5	305		
Noi	BX	305	R12	R25

Varianta 3 (4 tabele)

Masina 31 (similar se definesc Masina_32, Masina_33, Masina_34)..

Persoana	Vehicul
Eu	R25
Tu	205
El	R5
Noi	BX

Masina_34

Persoana	Vehicul
Noi	R25

Forma normală 2 (FN2)

O relaţie *R* este în a doua formă normală dacă şi numai dacă:

- relaţia *R* este în FN1;
- fiecare atribut care nu este cheie (nu participă la cheia primară) este dependent de întreaga cheie primară.

atasat_la

Cod_salariat#	Job_cod	Nr_proiect#	Functia	Suma
S1	Programator	P1	Supervizor	60
S1	Programator	P2	Cercetator	25
S1	Programator	P3	Auxiliar	10
S3	Vanzator	P3	Supervizor	60
S5	Inginer	P3	Supervizor	60

atasat_2a

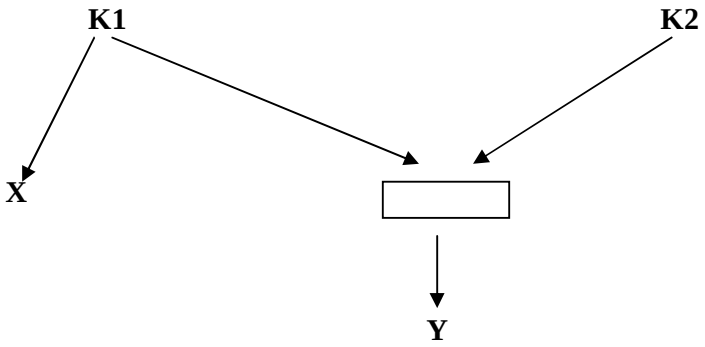
Cod_salariat#	Nr_proiect#	Functia	Suma
S1	P1	Supervizor	60
S1	P2	Cercetator	25
S1	P3	Auxiliar	10
S3	P3	Supervizor	60
S5	P3	Supervizor	60

atasat_2b

Cod_salariat#	Job_cod
S1	Programator
S3	Vanzator
S5	Inginer

A doua condiţie exprimă necesitatea total dependenţei de cheia primară. Această formă normală interzice manifestarea unor dependenţe funcţionale parţiale în cadrul relaţiei *R*!!!

Pentru a obţine o relaţie FN2 se poate aplica regula Casey-Delobel. Fie relaţia *R*(*K*1, *K*2, *X*, *Y*), unde *K*1 şi *K*2 definesc cheia primară, iar *X* şi *Y* sunt mulţimi de atribute, astfel încât $K1 \rightarrow X$. Din cauza dependenţei funcţionale $K1 \rightarrow X$ care arată că *R* nu este în FN2, se înlocuieşte *R* (fără pierdere de informaţie) prin două proiecţii *R*1(*K*1, *K*2, *Y*) şi *R*2(*K*1, *X*).



***Exemplu*.** Presupunem că un şantier poate executa mai multe lucrări de bază şi că o lucrare poate fi executată de mai multe şantiere.

LUCRARE(cod_obiectiv#, cod_lucrare#, nume);

SANTIER(nr_santier#, specialitate, sef);

EXECUTA(cod_obiectiv#, cod_lucrare#, nr_santier#, descriere, functie, conducator, data_inceput, data_sfarsit).

Pentru relaţia EXECUTA sunt evidente dependenţele:

{cod_obiectiv#, cod_lucrare#} $\rightarrow$ {data_inceput, data_sfarsit},

{cod_obiectiv#, cod_lucrare#, nr_santier#} $\rightarrow$ {descriere, functie, conducator}.

Relaţia EXECUTA este în FN1, dar nu este în FN2 deoarece atributele *data_inceput* şi *data_sfarsit* nu depind de numărul şantierului, deci nu depind de întreaga cheie primară. Pentru a obţine o relaţie în FN2 se aplică regula Casey Delobel şi relaţia EXECUTA se desface în:

EXECUTA_1(cod_obiectiv#, cod_lucrare#, nr_santier#, descriere, functie, conducator)

EXECUTA_2(cod_obiectiv#, cod_lucrare#, data_inceput, data_sfarsit).

Forma normală 3 (FN3)

Intuitiv, o relaţie *R* este în a treia formă normală dacă şi numai dacă:

- relaţia *R* este în FN2;
- fiecare atribut care nu este cheie (nu participă la o cheie) depinde direct de cheia primară.

Fie *R* o relaţie, *X* o submulţime de atribute ale lui *R* şi *A* un atribut al relaţiei *R*. *A* este **dependent tranzitiv** de *X* dacă există *Y* astfel încât $X \rightarrow Y$ şi $Y \rightarrow A$ (*A* nu aparţine lui *Y* şi *Y* nu determină pe *X*). *X* nu este dependent funcţional de *Y* sau *A*!

De exemplu, dacă $K_1, K_2, K_3 \rightarrow A_1$ şi dacă $K_1, K_2, A_1 \rightarrow A_2$, atunci $K_1, K_2, K_3 \rightarrow K_1, K_2, A_1$ şi $K_1, K_2, A_1 \rightarrow A_2$. Prin urmare, A_2 este dependent tranzitiv de K_1, K_2, K_3.

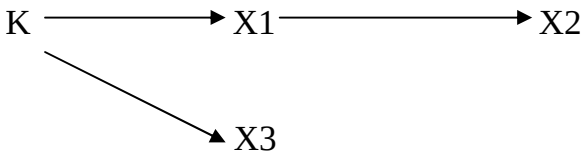
Formal, o relaţie *R* este în a treia formă normală dacă şi numai dacă:

- relaţia *R* este în FN2;
- fiecare atribut care nu este cheie (nu participă la o cheie) nu este dependent tranzitiv de nici o cheie a lui *R*.

O relaţie este în FN3 dacă şi numai dacă fiecare atribut (coloană) care nu este cheie, depinde de cheie, de întreaga cheie şi numai de cheie.

Pentru a obţine o relaţie FN3 se poate aplica regula Casey-Delobel.

Fie relaţia $R(K, X_1, X_2, X_3)$, unde atributul X_2 depinde tranzitiv de *K*, iar *K* este cheia primară a lui *R*. Presupunem că $K \rightarrow X_1 \rightarrow X_2$. Din cauza dependenţei funcţionale $X_1 \rightarrow X_2$ care arată că *R* nu este în FN3, se înlocuieşte *R* (fără pierdere de informaţie) prin două proiecţii $R1(K, X_1, X_3)$ şi $R2(X_1, X_2)$.



Exemplu: Tabelul *atasat_2a* nu este in FN3. De ce?

atasat_3a

Cod_salariat#	Nr_proiect#	Functia
S1	P1	Supervizor
S1	P2	Cercetator
S1	P3	Auxiliar
S3	P3	Supervizor
S5	P3	Supervizor

atasat_3b

Functia	Suma
Supervizor	60
Cercetator	25
Auxiliar	10

Exemplu. În tabelul EXECUTA1(cod_obiectiv#, cod_lucrare#, nr_santier#, descriere, functie, conducator) continuă să existe redundanţă în date.

Atributul *conducator* depinde indirect de cheia primară prin intermediul atributului *functie*. Între atributele relaţiei există dependenţele:

{cod_obiectiv#, cod_lucrare#, nr_santier#} → {descriere},

{cod_obiectiv#, cod_lucrare#, nr_santier#} → {functie} → {conducator}.

Pentru a aduce relaţia EXECUTA_1 în FN3 se aplică regula Casey-Delobel. Relaţia se desface, prin eliminarea dependenţelor funcţionale tranzitive, în proiecţiile:

EXECUTA11(cod_obiectiv#, cod_lucrare#, nr_santier#, descriere, functie)

EXECUTA12(functie, conducator).

Schema de sinteză pentru obţinerea lui FN3

Algoritmul de sinteză construieşte o acoperire minimală *F* a dependenţelor funcţionale totale. Se elimină atributele şi dependenţele funcţionale redundante. Mulţimea *F* este partiţionată în grupuri F_i, astfel încât în fiecare grup F_i sunt dependenţe funcţionale care au acelaşi membru stâng şi nu există două grupuri având acelaşi membru stâng. Fiecare grup F_i produce o schemă FN3. Algoritmul realizează o descompunere ce conservă dependenţele.

Algoritm SNF3 (aducerea unei relaţii în FN3 prin utilizarea unei scheme de sinteză):

1. Se determină *F* o acoperire minimală a lui *E* (mulţimea dependenţelor funcţionale).
2. Se descompune mulţimea *F* în grupuri notate F_i, astfel încât în cadrul fiecărui grup să existe dependenţe funcţionale având aceeaşi parte stângă.
3. Se determină perechile de chei echivalente (*X*, *Y*) în raport cu *F* (două mulţimi de atribute X, Y sunt chei echivalente dacă în mulţimea de dependenţe E există atât dependenţa X →Y, cât şi dependenta Y → X).

4. Pentru fiecare pereche de chei echivalente: se identifică grupurile F_i şi F_j care conţin dependenţele funcţionale cu partea stângă X şi respectiv Y; se formează un nou grup de dependenţe F_{ij}, care va conţine dependenţele funcţionale având membrul stâng (X, Y); se elimină grupurile F_i şi F_j, iar locul lor va fi luat de grupul F_{ij}.
5. Se determină o acoperire minimală a lui F, care va include toate dependenţele $X \rightarrow Y$, unde X şi Y sunt chei echivalente (celelalte dependenţe sunt redundante).
6. Se construiesc relaţii FN3 (câte o relaţie pentru fiecare grup de dependenţe funcţionale).

Se observă că algoritmul solicită

- determinarea unei acoperiri minimale (algoritmii EAR şi EDF);
- determinarea închiderii (A^+) unei mulţimi de atribute A în raport cu mulţimea de dependenţe funcţionale E (algoritm AIDF).

Determinarea acoperirii minimale presupune eliminarea atributelor şi dependenţelor redundante. Acoperirea minimală nu este unică şi depinde de ordinea în care sunt eliminate aceste atribute şi dependenţe redundante.

Algoritm EAR (elimină atributele redundante din determinantul dependenţelor funcţionale)

Pentru fiecare dependenţă funcţională din E şi pentru fiecare atribut din partea stângă a unei dependenţe funcţionale:

Pas1. Se elimină atributul considerat.

Pas2. Se calculează închiderea părţii stângi reduse.

Pas3. Dacă închiderea conţine toate atributele din determinantul dependenţei, atunci atributul eliminat la pasul 1 este redundant şi rămâne eliminat. În caz contrar, atributul nu este redundant şi se reintroduce în partea stângă a dependenţei funcţionale.

Algoritm EDF (elimină dependenţele funcţionale redundante din E)

Pentru fiecare dependenţă funcţională X →Y din E:

Pas1. Se elimină dependenţa din E.

Pas2. Se calculează închiderea X^+, în raport cu mulţimea redusă de dependenţe.

Pas3. Dacă Y este inclus în X^+, atunci dependenţa X →Y este redundantă şi rămâne eliminată. În caz contrar, se reintroduce în E.

Algoritm AIDF (determină închiderea lui A)

Pas1. Se caută dacă există în E dependenţe X →Y pentru care determinantul X este o submulţime a lui A, iar determinatul Y nu este inclus în A.

Pas2. Pentru fiecare astfel de dependenţă funcţională se adaugă mulţimii A atributele care constituie determinatul dependenţei.

Pas3. Dacă nu mai există dependenţe funcţionale de tipul de la pasul 1, atunci $A^+ = A$.

Exemplu:

Fie dependenţele funcţionale:

f1: F → N; f2: F → P; f3: P, F, N → U;

F4: P → C; f5: P → T; f6: C → T; f7: N → F.

Pas1 (suprimarea atributelor redundante). Atributul A_i este redundant în dependenţa funcţională $A_1, A_2, \ldots A_i, \ldots A_n \rightarrow Z$, dacă dependenţa funcţională $A_1, A_2, \ldots A_{i-1}, A_{i+1} \ldots A_n \rightarrow Z$ poate fi generată plecând de la mulţimea iniţială de dependenţe (E) şi de la axiomele considerate.

f1: F → N; f3: P, F, N → U sau N, P, F → U;

Aplicând axioma de pseudotranzitivitate se obţine:

F, P, F → U ➔ P, F → U

Pas2 (suprimarea dependenţelor funcţionale redundante). Dependenţa funcţională f este redundantă în E dacă $E^+ = (E - f)^+$, unde E^+ reprezintă închiderea lui E.

Se observă că f5 este redundantă (poate fi obţinută din f4 şi f6). La sfârşitul etapei se obţine:

f1: F → N; f2: F → P; f3: P, F → U; P este redundant =>F-->U

f4: P → C; f6: C → T; f7: N → F.

Pas3 (gruparea dependenţelor având acelaşi membru stâng).

F1 = {f1, f2, f3}; F2 = {f4}; F3 = {f6}; F4 = {f7}

Pas4 (regruparea mulţimilor F_i si F_j dacă există dependenţe de forma X → Y şi Y → X, unde X reprezinta partea stanga a dependenţei lui F_i şi Y este partea stângă a dependenţei lui F_j.

Din F1 şi F4 se obţine:

G1 = {f1, f2, f3, f7}; G2 = {f4}; G3 = {f6}.

Pas5 (generarea relaţiilor FN3).

R1(F#, N, P, U); R2(P#, C); R3(C#, T).

De remarcat: R1 nu este in BCNF! De ce? Exista dependenta N -->F.

Exerciţiu:

Presupunem că o parte din activităţiile de pe un aeroport sunt caracterizate de atributele:

A -- număr zbor;
B -- tip avion;
C -- aeroport plecare;
D -- aeroport sosire;
E -- linia aeriană;
F -- ora plecării;
G -- capacitate;
H -- număr locuri rezervate;
I -- data;
J -- tarif;
K -- prestaţii la bord.

Între aceste atribute există legături exprimate prin dependenţele:

A → BEFG
ACDI → H
CD → J
CDF → K
B → G
CF → ABE
AC → D
ABD → C
EG → B

Aplicând algoritmul de sinteză se obţin relaţiile:

R1(B#, G)
R2(E#, G#, B)
R3(A#, B, E, F)
R4(C#, D#, J)
R5(A#, C#, I#, H)
R6(A, C, D, F, K) în care cheile primare pot să fie oricare dintre:
AD sau AC sau CF.

Încercaţi să justificaţi acest rezultat!!!

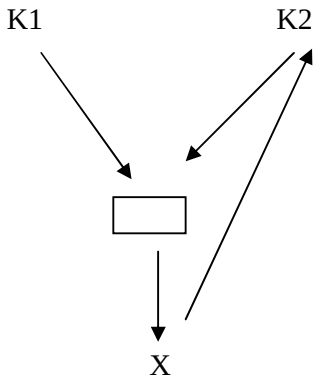
Forma normală Boyce-Codd (BCNF)

Determinantul este un atribut sau o mulţime de atribute neredundante, care constituie un identificator unic pentru alt atribut sau altă mulţime de atribute ale unei relaţii date.

Intuitiv, o relaţie *R* este în forma normală Boyce-Codd dacă şi numai dacă fiecare determinant este o cheie candidat.

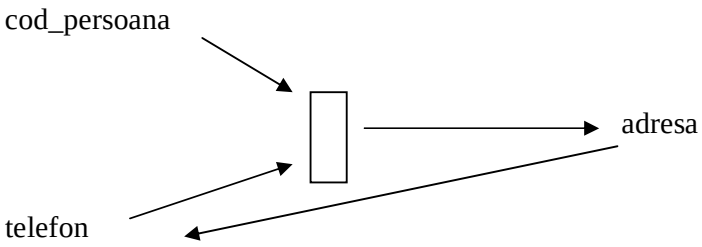
Formal, o relaţie *R* este în forma normală Boyce-Codd dacă şi numai dacă pentru orice dependenţă funcţională totală $X \rightarrow A$, *X* este o cheie (candidat) a lui *R*.

Regula Casey Delobel pentru R(K1#, K2#, X) presupunând că există dependenţa: X → K2. → R1(K1#, X) şi R2(X#, K2)



Exemplu:

ADRESA(cod_parsoana#, telefon#, adresa)



În dependenţa *adresa → telefon* se observă că determinantul nu este o cheie candidat. Relaţia ADRESA se desface în:

ADRESA_1(cod_persoana#, adresa);

ADRESA_2(adresa#, telefon).

Relaţiile sunt în BCNF, se conservă datele, dar nu se conservă dependenţele (s-a pierdut *cod_persoana, telefon → adresa*).

Exemplu:

Relaţia INVESTESTE_IN leagă entităţile INVESTITOR şi OBIECTIV_INVESTITIE. Ea are schema relaţională:

INVESTESTE_IN(cod_contractant#, cod_obiectiv#, nr_contract, cota_parte). Între atributele relaţiei există dependenţele:

{cod_contractant#, cod_obiectiv#} $\rightarrow$ {nr_contract, cota_parte},
{nr_contract} $\rightarrow$ {cod_obiectiv}.

Se aplică regula Casey-Delobel şi se aduce relaţia în BCNF.

INVESTESTE_IN_1(cod_obiectiv, nr_contract#);
INVESTESTE_IN_2(cod_contractant#, nr_contract, cota_parte).

Pentru ca o relaţie să fie adusă în BCNF nu trebuie în mod obligatoriu să fie în FN3. Se pot aduce în BCNF şi relaţii aflate în FN1 sau FN2. Acest lucru este posibil întrucât dependenţele funcţionale parţiale şi cele tranzitive sunt tot dependenţe noncheie, adică dependenţe ai căror determinanţi nu sunt chei candidat. Presupunem că *R* este o relaţie ce conţine atributele *A*.

Algoritm TFBCNF (aducerea unei relaţii *R* din FN1 în BCNF)

1. Dacă relaţia conţine cel mult două atribute, atunci *R* este în BCNF şi algoritmul s-a terminat.
2. Dacă relaţia conţine mai mult de două atribute, se consideră toate perechile (*X*, *Y*) de atribute distincte din *A*.
3. Se determină A_1^+, închiderea mulţimii $A_1 = A - \{X, Y\}$.
4. Dacă pentru orice pereche (*X*, *Y*), $X \notin A_1^+$ atunci relaţia *R* este în BCNF şi algoritmul s-a terminat.
5. În caz contrar (pentru cel puţin o pereche (*X*, *Y*), *X* aparţine lui A_1^+), relaţia *R* nu este în BCNF.
6. Se reduce progresiv schema relaţiei şi se reia algoritmul, exploatând relaţia redusă. Orice relaţie obţinută prin reducerea lui *R* şi care este în BCNF se consideră ca făcând parte din descompunerea lui *R* în procesul aducerii sale în BCNF.

Forma normală 4 (FN4)

FN4 elimină redundanţele datorate relaţiilor *m*:*n*, adică datorate dependenţei multiple.

Intuitiv, o relaţie *R* este în a patra formă normală dacă şi numai dacă relaţia este în BCNF şi nu conţine relaţii *m*:*n* independente.

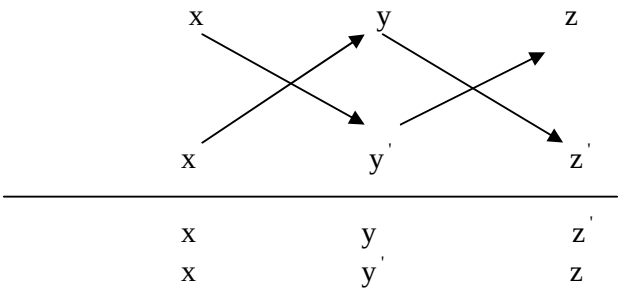
Fie R o relaţie definită pe o mulţime de atribute $A = \{A_1, A_2, ..., A_n\}$ şi fie $X, Y, Z \subset A$. Se spune că X **multidetermină** pe Z sau că Z este multidependent de X :

- dacă pentru fiecare valoare a lui Z în R există numai o valoare pentru perechea (X, Y);
- dacă valoarea lui Z depinde numai de valoarea lui X.

Acest tip de dependenţă, numită şi multivaloare sau multidependenţă (MVD) se notează prin $X \rightarrow\rightarrow Z$.

Intuitiv, multidependenţa reprezintă situaţia în care valoarea unui atribut (sau a unei mulţimi de atribute) determină o mulţime de valori a altui atribut (sau mulţimi de atribute)!!!

Multidependenţa $X \rightarrow\rightarrow Y$ poate fi gândită ca o regulă de deducţie: dacă tuplurile $<x, y, z>$ şi $<x, y', z'>$ sunt în relaţie la un moment r, atunci la momentul r sunt în relaţie şi tuplurile $<x, y, z'>$ şi $<x, y', z>$.



Orice dependenţă funcţională este o multidependenţă. Afirmaţia inversă nu este adevărată. Dacă $X \rightarrow Y$ (FD), atunci pentru oricare două tupluri $<x, y, z>$ şi $<x, y', z'>$, se obţine $y = y'$. Prin urmare în relaţie apar tuplurile $<x, y', z>$ şi $<x, y, z'>$ şi deci $X \rightarrow\rightarrow Y$ (MVD).

Fie W, V, X, Y şi Z submulţimi de atribute ale unei scheme relaţionale R. Fiind dată o mulţime T de multidependenţe există o mulţime completă de axiome (Ax1–Ax8) care permit obţinerea tuturor multidependenţelor ce se pot deduce din mulţimea T:

Ax1. Dacă $Y \subset X$, atunci $X \rightarrow Y$.

Ax2. Dacă $X \rightarrow Y$, atunci $X \cup Z \rightarrow Y \cup Z$.

Ax3. Dacă $X \rightarrow Y$ şi $Y \rightarrow Z$, atunci $X \rightarrow Z$.

Ax4. Dacă $X \rightarrow\rightarrow Y$, atunci $X \rightarrow\rightarrow R - \{X \cup Y\}$.

Ax5. Dacă $X \rightarrow\rightarrow Y$ şi $V \subset W$, atunci $W \cup X \rightarrow\rightarrow V \cup Y$.

Ax6. Dacă $X \rightarrow\rightarrow Y$ şi $Y \rightarrow\rightarrow Z$, atunci $X \rightarrow\rightarrow (Z - Y)$.

Ax7. Dacă $X \rightarrow Y$, atunci $X \rightarrow\rightarrow Y$.

Ax8. Dacă $X \rightarrow\rightarrow Y$, $Z \rightarrow W$, $W \subset Y$ şi $Y \cap Z = \varnothing$, atunci $X \rightarrow W$.

O **multidependenţă elementară** este o multidependenţă care are părţi stângi şi drepte minimale (nu există $X' \subset X$ şi $Y' \subset Y$ a.i. $X' \rightarrow\rightarrow Y'$).

Formal, relaţia R este în a patra formă normală dacă şi numai dacă:

- R este în BCNF;
- orice dependenţă multivaloare este o dependenţă funcţională.

O relaţie BCNF este în FN4 dacă pentru orice multidependenţă elementară de forma $X \rightarrow\rightarrow Y$, X este o supercheie a lui R. Aducerea relaţiilor în FN4 presupune eliminarea dependenţelor multivaloare atunci când sunt mai mult de una în cadrul unei relaţii.

Regula de descompunere în relaţii FN4. Fie $R(X, Y, Z)$ o schemă relaţională care nu este în FN4 şi fie $X \rightarrow\rightarrow Y$ o multidependenţă elementară care nu este de forma „CHEIE $\rightarrow\rightarrow$ atribut“. Această relaţie este descompusă prin proiecţie în două relaţii:

$$R = \text{JOIN}(\Pi_{X \cup Y}(R), \Pi_{X \cup Z}(R)).$$

Aplicând recursiv această regulă, se obţin relaţii FN4.

***Exemplu*.** Fie relaţia INVESTITIE(cod_contractant#, denumire, telefon) şi presupunem că un investitor poate avea mai multe numere de telefon şi că poate investi în mai multe obiective. Între atributele relaţiei există multidependenţele:

cod_contractant# $\rightarrow\rightarrow$ denumire;

cod_contractant# $\rightarrow\rightarrow$ telefon.

Relaţia INVESTITIE este în BCNF. Pentru a aduce relaţia în FN4 o vom descompune prin proiecţie în două relaţii:

INVESTITIE_1(cod_contractant#, denumire),

INVESTITIE_2(cod_contractant#, telefon).

INVESTITIE = JOIN(INVESTITIE_1, INVESTITIE_2).

Forma normală 5 (FN5)

FN5 îşi propune eliminarea redundanţelor care apar în relaţii *m:n* dependente. În general, aceste relaţii nu pot fi descompuse. S-a arătat că o relaţie de tip 3 este diferită de trei relaţii de tip 2. Există totuşi o excepţie, şi anume, dacă relaţia este ciclică

Intuitiv, o relaţie *R* este în forma normală 5 dacă şi numai dacă:

1. relaţia este în FN4;
2. nu conţine dependenţe ciclice.

Dependenţa funcţională şi multidependenţa permit descompunerea prin proiecţie, fără pierdere de informaţie, a unei relaţii în două relaţii. Regulile de descompunere (FN1 – FN4) nu dau toate descompunerile posibile prin proiecţie ale unei relaţii. Există relaţii care nu pot fi descompuse în două relaţii dar pot fi descompuse în trei, patru sau mai multe relaţii fără a pierde informaţii. Pentru a obţine descompuneri L-*join* în trei sau mai multe relaţii, s-a introdus conceptul de *join*-dependenţă sau dependenţă la compunere (JD).

Fie $\{R_1, R_2, ..., R_p\}$ o mulţime de scheme relaţionale care nu sunt disjuncte şi a căror reuniune este *R*.

R satisface ***join*-dependenţa** $*\{R_1, R_2, ..., R_p\}$ dacă la fiecare moment al lui *R* are loc egalitatea:

$$R = \text{JOIN}(\Pi_{\alpha 1}(R), \Pi_{\alpha 2}(R), ..., \Pi_{\alpha p}(R))$$

unde α_k reprezintă mulţimea atributelor corespunzătoare lui $R_k (1 \le k \le p)$.

Join-dependenţa $*\{R_1, R_2, ..., R_p\}$ are loc în *R*, dacă $R_1, R_2, ..., R_p$ este o descompunere L-*join* a lui *R*. Pentru $p = 2$ se regăseşte multidependenţa. O *join*-dependenţă $*\{R_1, R_2, ..., R_p\}$ în care una dintre R_i este chiar *R*, defineşte o *join*-dependenţă trivială.

Join-dependenţa generalizează multidependenţa. Într-adevăr, multidependenţa $X \rightarrow\rightarrow Y$ în relaţia $R(X, Y, Z)$ (deci şi $X \rightarrow\rightarrow Z$), corespunde *join*-dependenţei $*\{X \cup Y, X \cup Z\}$. Invers, *join*-dependenţa $*\{R_1, R_2\}$ corespunde multidependenţei $R_1 \cap R_2 \rightarrow\rightarrow R_1 - (R_1 \cap R_2)$.

Formal, o relaţie *R* este în FN5 dacă şi numai dacă orice *join*-dependenţă $*\{R_1, R_2, ..., R_p\}$ care are loc în *R* fie este trivială, fie conţine o supercheie a lui *R* (adică, o anumită componentă R_i este o supercheie a lui *R*). Cu alte cuvinte, o relaţie *R* este în FN5 dacă orice *join*-dependenţă definită pe *R* este implicată de cheile candidat ale lui *R*.

Între mulţimile de atribute *X*, *Y* şi *Z* din cadrul relaţiei *R* există o *join*-dependenţă dacă există multidependenţe între fiecare dintre perechile de mulţimi (*X*, *Y*), (*Y*, *Z*) şi (*X*, *Z*).

Aducerea în FN5 prin eliminarea join dependenţelor!

Exemplu.

Fie schema R(furnizor, cod_consumabil, cantitate, pret).

Reguli:

- un furnizor produce mai multe consumabile;
- nu toţi furnizorii produc aceleaşi consumabile;
- preţul unui consumabil de la un furnizor este variabil şi nu depinde de cantitate.

Furnizor	Cod_consumabil	Cantitate	Pret
F1	1	500	100
F2	1	100	80
F2	1	500	100
F2	2	500	100

Relaţia este în FN4, dar există redundanţă în date. Relaţia se descompune prin proiecţie în:

R1(furnizor#, cod_consumabil#)

R2(furnizor#, cantitate, pret)

R3(cod_consumabil#, cantitate, pret).

S-au eliminat redundanţele:

(F2,1) pentru R1;

(F2, 500, 100) pentru R2;

(1, 500, 100) pentru R3.

Se observă că:
JOIN(R1, R2) ≠ R;
JOIN(R1, R3) ≠ R;
JOIN(R3, R2) ≠ R;
JOIN(R1, R2, R3) = JOIN(R1, JOIN(R2, R3)) = R

Există join dependenţa:

*{R1(furnizor, cod_consumabil), R2(furnizor, cantitate, pret), R3(cod_consumabil, cantitate, pret)}

Exemplu.

Fie schema relaţională:

EXECUTANT(nr_santier#, cod_obiectiv#, cod_lucrare#, data_inceput, data_sfarsit). Un şantier poate executa mai multe lucrări referitoare la acelaşi obiectiv sau poate executa o lucrare pentru un obiectiv în intervale de timp distincte. Se presupune că mai multe şantiere pot executa aceeaşi lucrare, în acelaşi interval de timp sau în intervale de timp distincte.

Relaţia, datorită dependenţelor formulate anterior, nu este în FN5. Ea se poate desface prin proiecţie în trei relaţii:

EX1(nr_santier#, cod_obiectiv#, cod_lucrare#);

EX2(nr_santier#, data_inceput, data_sfarsit);

EX3(cod_obiectiv#, cod_lucrare#, data_inceput, data_sfarsit).

Sunt evidente relaţiile:

EXECUTANT ≠ JOIN(EX1, EX2),

EXECUTANT ≠ JOIN(EX1, EX3),

EXECUTANT ≠ JOIN(EX2, EX3),

EXECUTANT = JOIN(JOIN(EX1, EX2), EX3).

Concluzii:

1. FN1 → FN2 elimină redundanţele datorate dependenţei netotale a atributelor care nu participă la o cheie, faţă de cheile lui *R*. Se suprimă dependenţele funcţionale care nu sunt totale.
2. FN2 → FN3 elimină redundanţele datorate dependenţei tranzitive. Se suprimă dependenţele funcţionale tranzitive.
3. FN3 → BCNF elimină redundanţele datorate dependenţei funcţionale. Se suprimă dependenţele în care partea stângă nu este o supercheie.
4. BCNF → FN4 elimină redundanţele datorate multidependenţei. Se suprimă toate multidependenţele care nu sunt şi dependenţe funcţionale.
5. FN4 → FN5 elimină redundanţele datorate dependentei. ciclice. Se suprimă toate *join*-dependenţele care nu sunt implicate de o cheie.
6. BCNF, FN4 şi FN5 corespund la regula că orice determinant este o cheie, dar de fiecare dată dependenţa cu care se defineşte determinantul este alta şi anume dependenţa funcţională, multidependenţa sau *join*-dependenţa).
7. Descompunerea unei relaţii FN2 în FN3 conservă datele şi dependenţele, pe când descompunerea unei relaţii FN3 în BCNF şi, respectiv, a unei relaţii BCNF în FN4 conservă doar datele.