Curs 1: Prezentare

Baze de date

Silviu Vasile
vsl@fmi.unibuc.ro

Informatii generale:

- Program:
  - Curs: 14:00 – 17:00 (miercuri)
- Forma de evaluare:
  - Examen + Proiect
- Cursurile vor contine notiunile necesare pentru rezolvarea exercitiilor din laborator – recomand sa folositi SQLDeveloper-ul in timpul cursului

- Pauza
  - 2x10 minute
- Contact:
  - email: vsl@fmi.unibuc.ro
  - web: http://193.226.51.37

Examen

- Proiect 30% (tema aleasa pana la 03.11.2021) – obligatoriu
- Laborator – obligatoriu (nu se face prezenta)

- Examen 70%

- Promovare - doar daca la oricare dintre cele 2 componente se obtin note de promovare

Schema succinta a cursului:

- Generalitati despre baze de date
- Arhitectura sistemelor de gestiune a bazelor de date
- *Design*-ul bazelor de date
- Diagrame entitate relatie
- Proiectarea modelului relational
- LMD, LDD, LCD – sintaxa *SQL*

Introducere:

- Ce pondere considerati ca au bazele de date in viata de zi cu zi?

Introducere:

- Ati folosit astazi o baza de date? La ce?

Introducere:

- Ce este o baza de date?

Introducere:

- Ce functionalitati trebuie sa indeplineasca o baza de date?

Introducere:

- Ce deosebiri considerati ca exista intre un tabel al unei baze de date si un «sac»?

Introducere:

- Ce este un SGBD? Dar SQL?

Introducere:

- Ce exemple de SGBD-uri cunoasteti?

Introducere:

- Ce este o baza de date relationala?

Introducere:

- De ce considerati ca intr-o baza de date sunt necesare mai multe tabele?

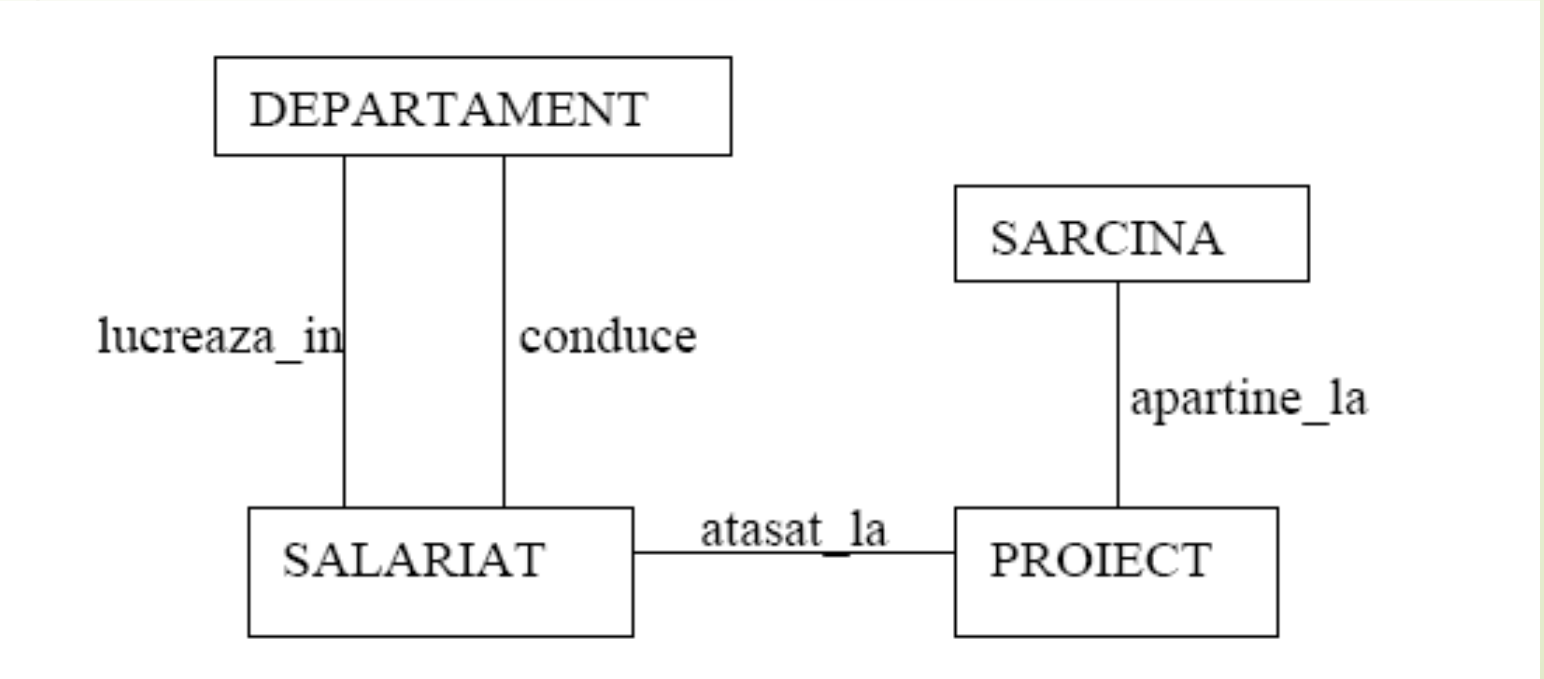
Bibliografie:

- **Modelarea bazelor de date, Ileana Popescu, 2001**
- **Oracle SQL By Example (4th Edition) Paperback – August 22, 2009**
- **Oracle Database 12c SQL Paperback – August 20, 2013**
- **Murach's Oracle SQL and PL/SQL for Developers, 2nd Edition**
- **Oracle PL/SQL Programming Paperback – February 16, 2014**
- **OCA Oracle Database 12c SQL Fundamentals I Exam Guide (Exam 1Z0-061) (Oracle Press)**
- **The Language of SQL: How to Access Data in Relational Databases Paperback – June 3, 2010**
- **Expert Oracle SQL: Optimization, Deployment, and Statistics Paperback – June 24, 2014**

Diagrame entitate-relație

- Diagrama E/R – model neformalizat pentru reprezentarea unui sistem din lumea reală. Este un model de date conceptual de nivel înalt dezvoltat de Chen (1976) pentru a facilita proiectarea bazelor de date.
- Modelul de date conceptual este independent de:
  - tipul SGBD-ului;
  - platforma *hardware* utilizata.
- Modelul conceptual este constituit din concepte care descriu:
  - structura bazei de date;
  - tranzactii de regasire si reactualizare asociate.

- **Entitate**: persoană, loc, concept, activitate, eveniment care este



Observații:

- Entitățile devin tabele în modelele relaționale.
- În general, entitățile se scriu cu litere mari.
- Entitățile sunt substantive, dar nu orice substantiv este o entitate.
- Pentru fiecare entitate este obligatoriu să se dea o descriere detaliată.
- Nu pot exista, în aceeaşi diagramă, două entităţi cu acelaşi nume, sau o aceeaşi entitate cu nume diferite.

- **Cheia primară** este un identificator unic în cadrul entității, făcând distincție între valori diferite ale acesteia.
- Cheia primară:
  - trebuie să fie unică şi cunoscută la orice moment;
  - trebuie să fie controlată de administratorul bazei;
  - trebuie să nu conțină informații descriptive, să fie simplă, fără ambiguități;
  - să fie stabilă;
  - să fie familiară utilizatorului.

- **Relație** (asociere): o comunicare între două sau mai multe entități. Existența unei relații este subordonată existenței entităților pe care le leagă. Gradul (tipul) unei relații este dat de numărul entităților participante la relația respectivă.
- *Observații*:
  - În modelul relațional, relațiile devin tabele speciale sau coloane speciale care referă chei primare.
  - Relațiile sunt verbe, dar nu orice verb este o relație.
  - Pentru fiecare relație este important să se dea o descriere detaliată.
  - În aceeași diagramă pot exista relații diferite cu același nume. În acest caz, le diferențiază entitățile care sunt asociate prin relația respectivă.
  - Pentru fiecare relație trebuie stabilită cardinalitatea (maximă şi minimă) relației, adică numărul de tupluri ce aparțin relației.
  - poate (cardinalitate maximă) → trebuie (cardinalitate minima)

Exemplu:

- Câți salariați **pot** lucra într-un departament? Mulți!
- În câte departamente **poate** lucra un salariat? In cel mult unul!

➔ Relația SALARIAT_lucreaza_in_DEPARTAMENT are cardinalitatea maximă ***many-one*** (n:1).

Exemplu:

- Câți salariați **trebuie** să conducă un departament? Cel puțin unul!
- Câte departamente **trebuie** să conducă un salariat? Zero!

➔ Relația SALARIAT_conduce_DEPARTAMENT are cardinalitatea minimă ***one-zero*** (1:0).

- **Atribut**: proprietate descriptivă a unei entități sau a unei relații. Atributele pot fi simple, compuse, cu valori multiple, derivate.

Observații:

- Trebuie făcută distincția între tipul atributului (devine coloană în modelele relaționale) și valoarea acestuia (devine valoare în coloane).
- Atributele sunt substantive, dar nu orice substantiv este atribut.
- Fiecărui atribut trebuie să i se dea o descriere completă (exemple, contraexemple, caracteristici).
- Pentru fiecare atribut trebuie specificat numele, tipul fizic (*integer*, *float*, *char* etc.), valori posibile, valori implicite, reguli de validare, tipuri compuse.

- Pentru proiectarea **diagramei entitate-relație** au fost stabilite anumite reguli (nu sunt unice):
  - entitățile sunt reprezentate prin dreptunghiuri;
  - relațiile dintre entități sunt reprezentate prin arce neorientate;
  - atributele care reprezintă chei primare trebuie subliniate sau marcate prin simbolul „#“, plasat la sfârşitul numelui acestor atribute;
  - cardinalitatea minimă este indicată în paranteze, iar cardinalitatea maximă se scrie fără paranteze;
  - nu trebuie specificate toate atributele.

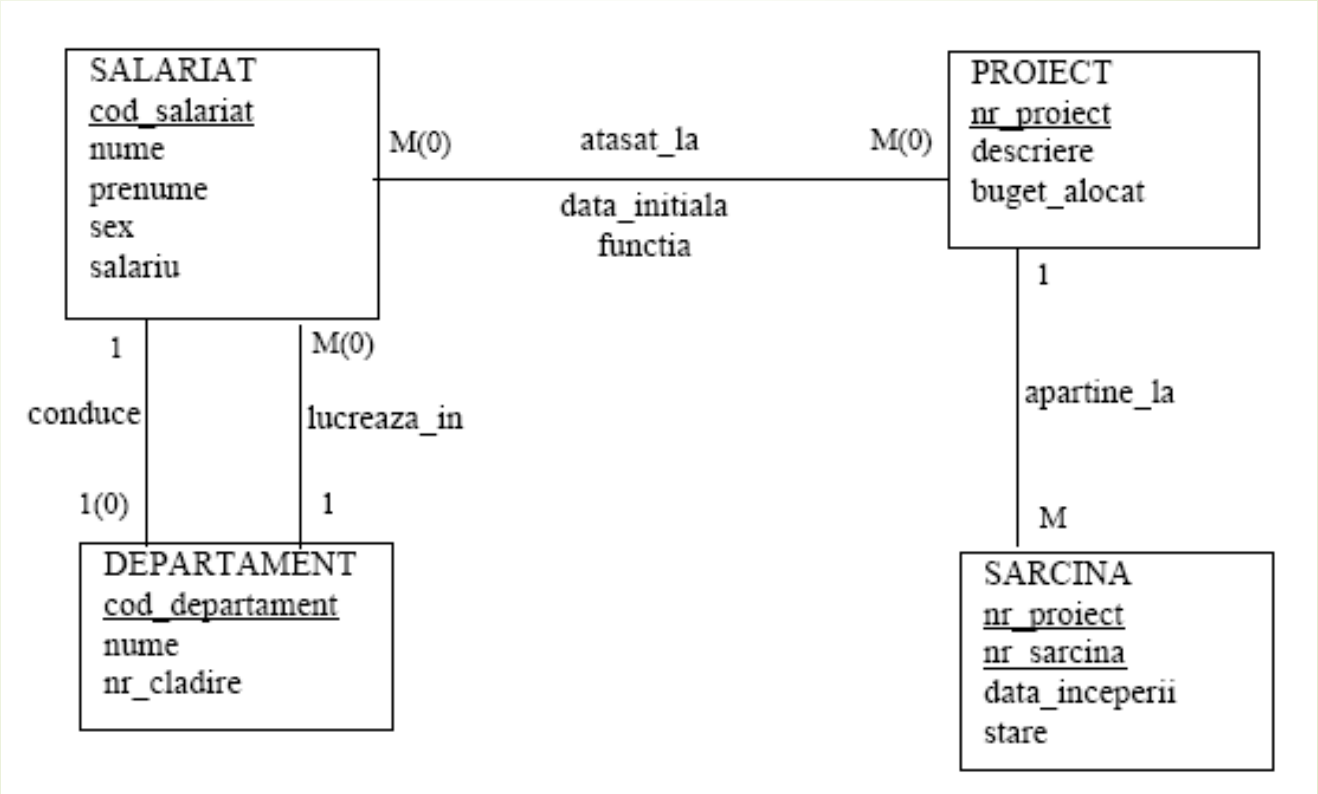

SALARIAT
cod_salariat
nume
prenume
sex
salariu
M(0)
atasat_la
M(0)
data_initiala
functia
PROIECT
nr_proiect
descriere
buget_alocat
1
conduce
1(0)
M(0)
lucreaza_in
1
1
apartine_la
M
DEPARTAMENT
cod_departament
nume
nr_cladire
SARCINA
nr_proiect
nr_sarcina
data_inceperii
stare

Diagrama E/R

Cazuri speciale de entități, relații, atribute și modul lor de reprezentare în cadrul diagramei entitate-relație.

1. **Entitate dependentă** – nu poate exista în mod independent (SARCINA depinde de PROIECT). Cheia primară a unei entități dependente include cheia primară a sursei (*nr_proiect*) și cel puțin o descriere a entității (*nr_sarcina*). Entitatea dependentă se desenează prin dreptunghiuri cu linii mai subțiri.
2. Moștenirea atributelor. **Subentitate** (subclasă) – submulțime a unei alte entități, numită **superentitate** (superclasă) (SALARIAT < — > PROGRAMATOR). Subentitatea se desenează prin dreptunghiuri incluse în superentitate. Există o relație între o subentitate și o superentitate, numită ISA, care are cardinalitatea maximă 1:1 și minimă 1:0. Cheile primare, atributele și relațiile unei superentități sunt valabile pentru orice subentitate. Afirmația reciprocă este falsă.

3. Generalizare. Din entități similare care au mai multe atribute comune se pot crea superentități. Aceste superentități conțin atributele comune, iar atributele speciale sunt asignate la subentități. Pentru noile superentități se introduc chei primare artificiale.
4. Specializare. După valorile unor atribute clasificatoare se pot determina **clase**. Un grup de subentități reciproc exclusive definește o clasă. Claselе se aliniază în desen vertical.
5. Într-o diagramă E/R se pot defini relații recursive.
6. Unele relații sunt relative la două entități și le numim de tip 2, iar dacă relațiile implică mai mult de două entități, le vom numi de tip 3. Trei relații de tip 2 sunt diferite de o relație de tip 3. Rupând o relație de tip 3 în trei relații de tip 2, pot apărea informații incorecte.

7. Trebuie excluse din model relaţiile indirecte deoarece ele pot conduce la redundanţă în baza de date.
8. Atributele derivabile trebuie eliminate şi introduse expresii prin care aceste atribute pot fi calculate.
9. Relaţie sau atribut? Dacă un atribut al unei entităţi reprezintă cheia primară a unei alte entităţi, atunci el reflectă o relaţie (*cod_departament* în tabelul SALARIAT).
10. Entitate sau relaţie? Se cercetează cheia primară. Dacă aceasta combină cheile primare a două entităţi, atunci este vorba de o relaţie. (cheia primară a relaţiei *asociat_la* combină *cod_salariat* cu *nr_proiect*, prin urmare, *SALARIAT_asociat la_PROIECT* va defini o relaţie şi nu o entitate).

11. Un atribut indirect este inoportun. El nu descrie real relația sau entitatea. Prin urmare, atributele indirecte trebuie reasignate. De fapt, un atribut indirect este un caz special de relație indirectă care trebuie eliminată pentru că introduce redundanță în date (numărul clădirii în care lucrează un salariat este un atribut al entității DEPARTAMENT şi nu este o caracteristică a entității SALARIAT).
12. Există atribute opționale, a căror valoare este uneori necunoscută, alteori neaplicabilă. Aceste atribute trebuie introduse la subentități (comisionul pentru deplasare şi zona de lucru sunt atribute specifice unui agent teritorial şi trebuie introduse la subentitatea AGENT_TERITORIAL).

Modelul EER (modelul E/R extins) = Diagrama E/R + concepte adiționale (subclasă, superclasă, moştenire, specializare, generalizare).

Algoritmul pentru proiectarea diagramei entitate-relație:

1. Identificarea entităților din cadrul sistemului analizat;
2. identificarea relațiilor dintre entități şi stabilirea cardinalității;
3. identificarea atributelor aferente entităților şi asocierilor dintre entități;
4. stabilirea atributelor de identificare a entităților (stabilirea cheilor).

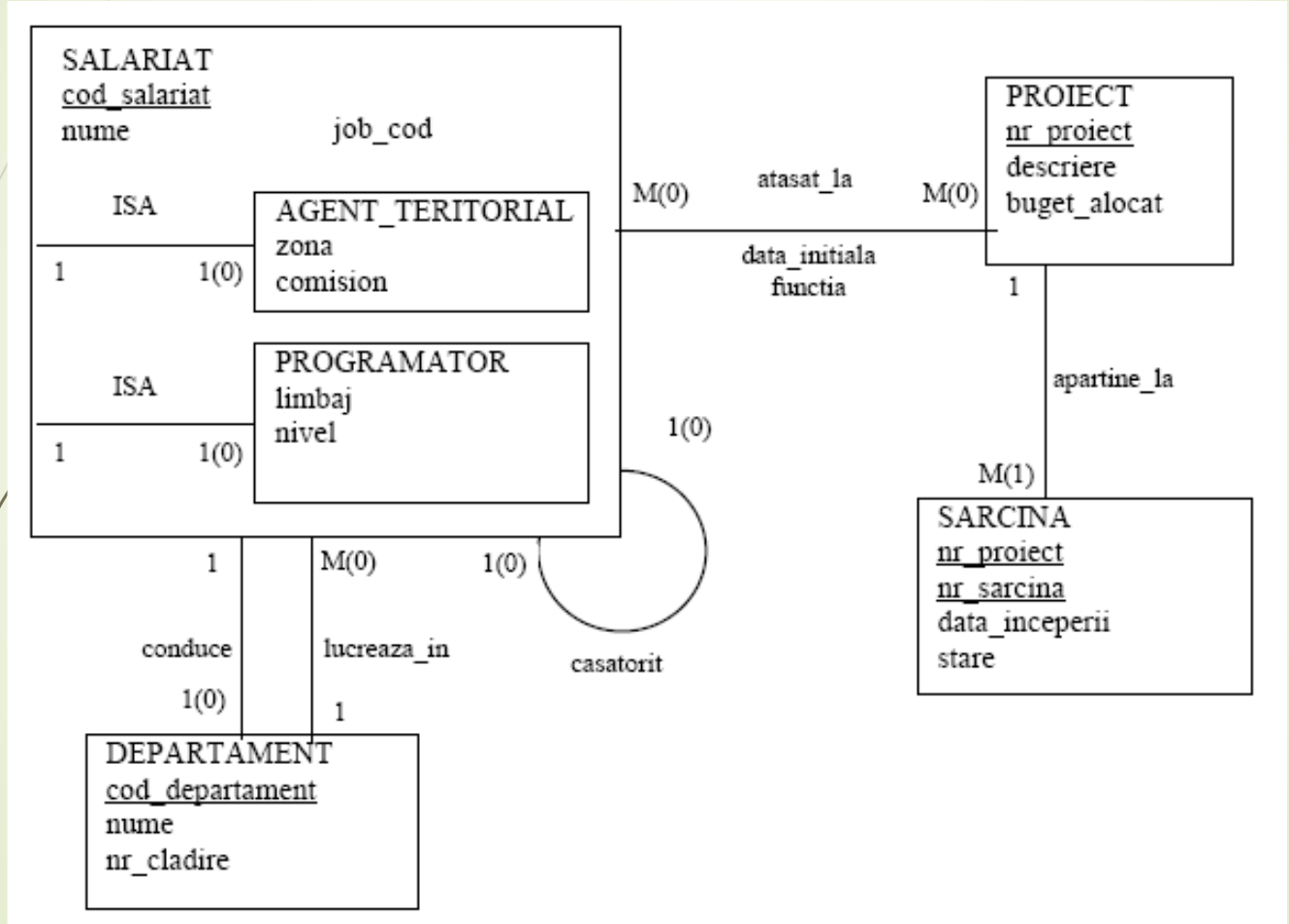
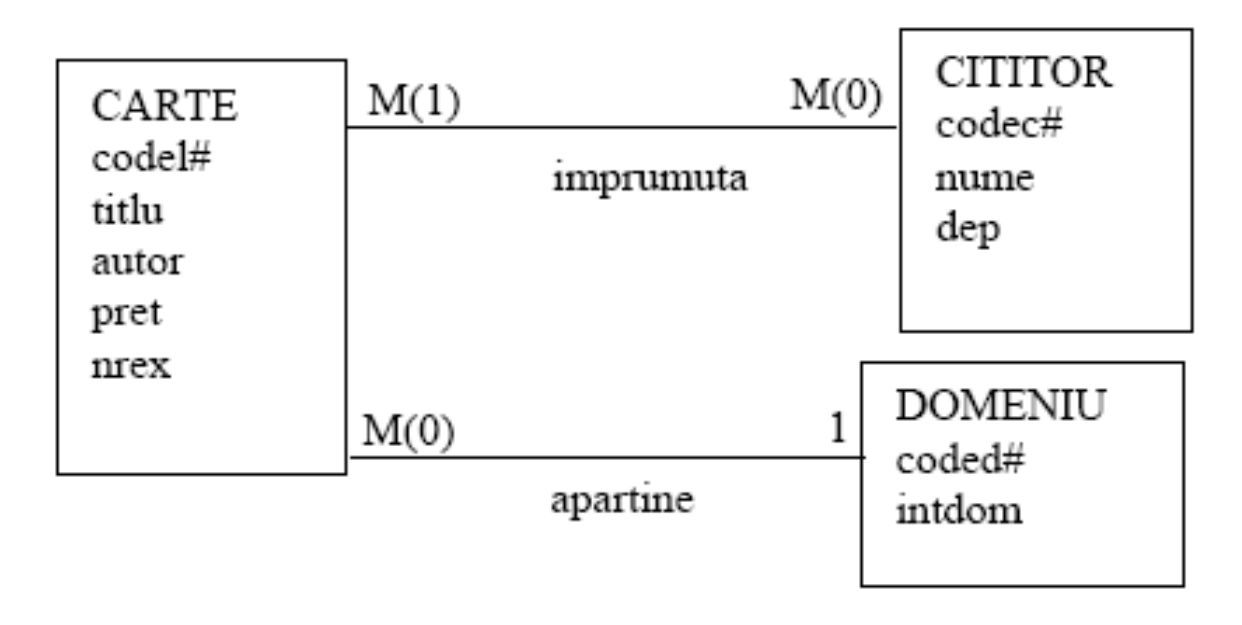

SALARIAT
cod_salariat
nume
job_cod
ISA
AGENT_TERITORIAL
zona
comision
1
1(0)
ISA
PROGRAMATOR
limbaj
nivel
1
1(0)
M(0)
atasat_la
M(0)
PROIECT
nr_proiect
descriere
buget_alocat
data_initiala
functia
1
apartine_la
M(1)
SARCINA
nr_proiect
nr_sarcina
data_inceperii
stare
1(0)
1
M(0)
1(0)
conduce
1(0)
lucreaza_in
1
casatorit
DEPARTAMENT
cod_departament
nume
nr_cladire

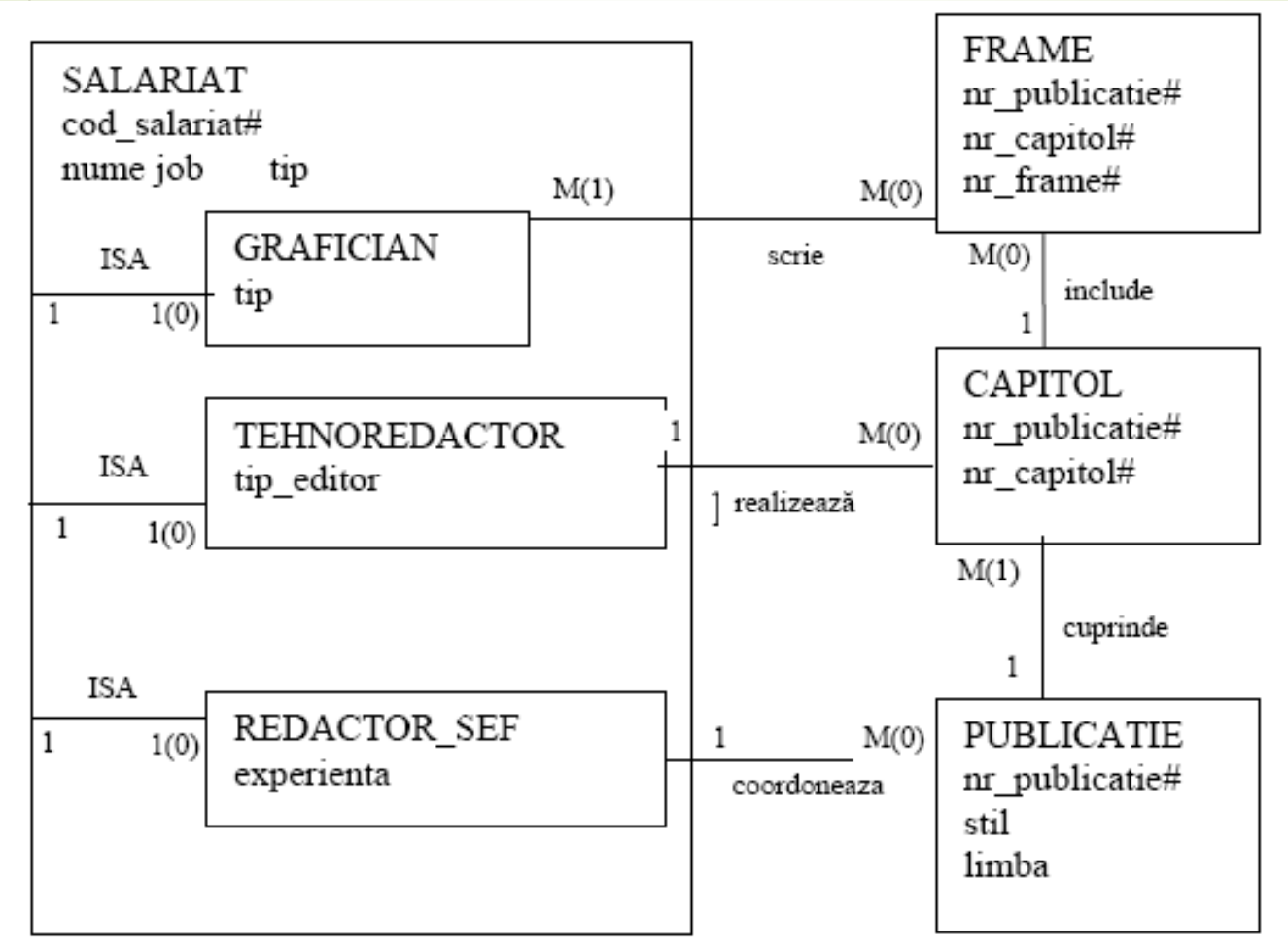
Diagrama E/R

Gestiunea activităților de împrumut dintr-o bibliotecă

- S-a presupus (restrictiv) că într-o zi un cititor nu poate împrumuta, de mai multe ori, aceeași carte. Modelul prezintă anomalii (de exemplu, cheia primară de la entitatea carte)! A fost gândit în această manieră cu scop pur didactic.
- **Entitățile** și **relațiile** care intervin în acest model sunt următoarele:
  - CARTE (entitate independentă) – orice carte care se găsește în inventarul bibliotecii. Cheia primară este atributul *codel*.
  - CITITOR (entitate independentă) – orice cititor care poate împrumuta cărți. Cheia primară este atributul *codec*.
  - DOMENIU (entitate independenta) – domeniul căruia îi aparține o carte. Cheia primara este atributul *coded*.
  - IMPRUMUTA – relație având cardinalitatea *m:m* care leagă entitățile CITITOR şi CARTE.
  - APARTINE – relație care leagă atributele CARTE şi DOMENIU. Relația are cardinalitatea maximă *m*:1, iar cardinalitatea minimă 1:1.

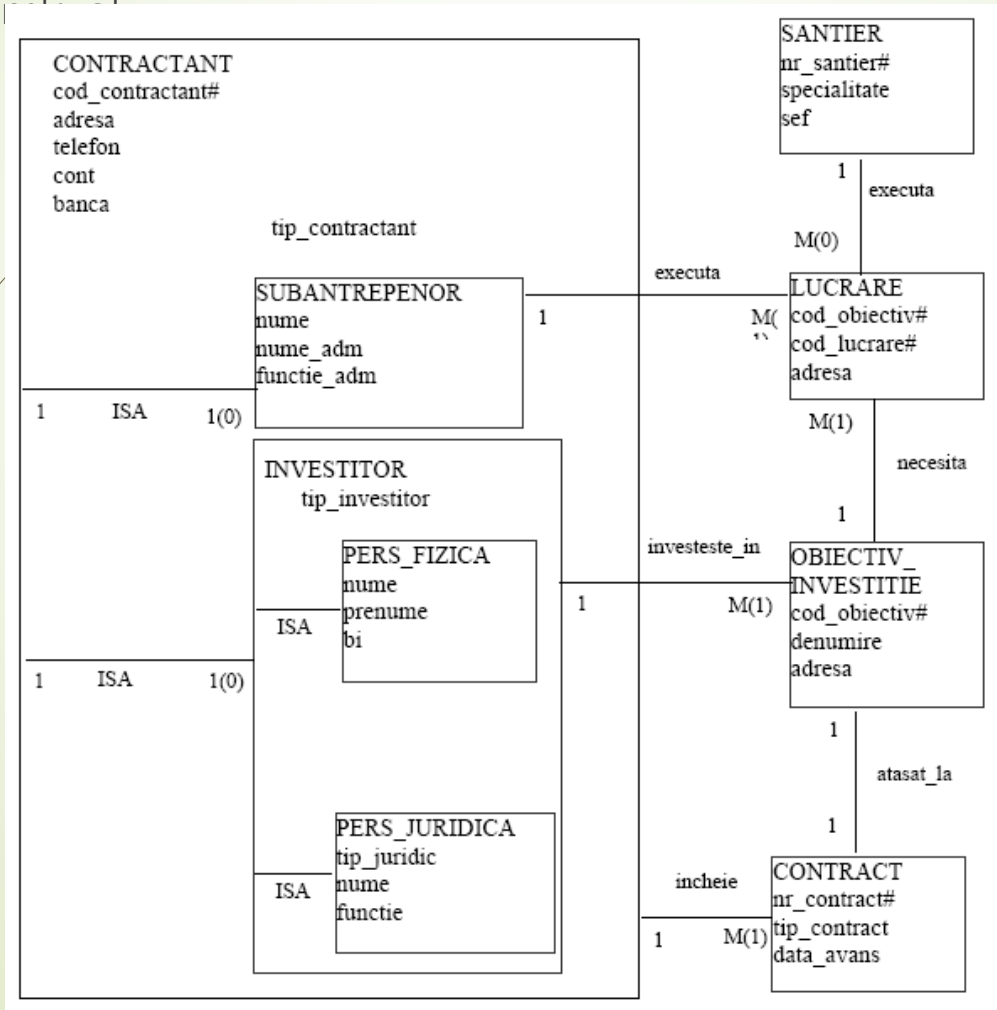

CARTE
codel#
titlu
autor
pret
nrex
M(1)
M(0)
imprumuta
CITITOR
codec#
nume
dep
M(0)
1
apartine
DOMENIU
coded#
intdom

Gestiunea activităților de editare dintr-o editură



Gestiunea activităților unei firme de construcții

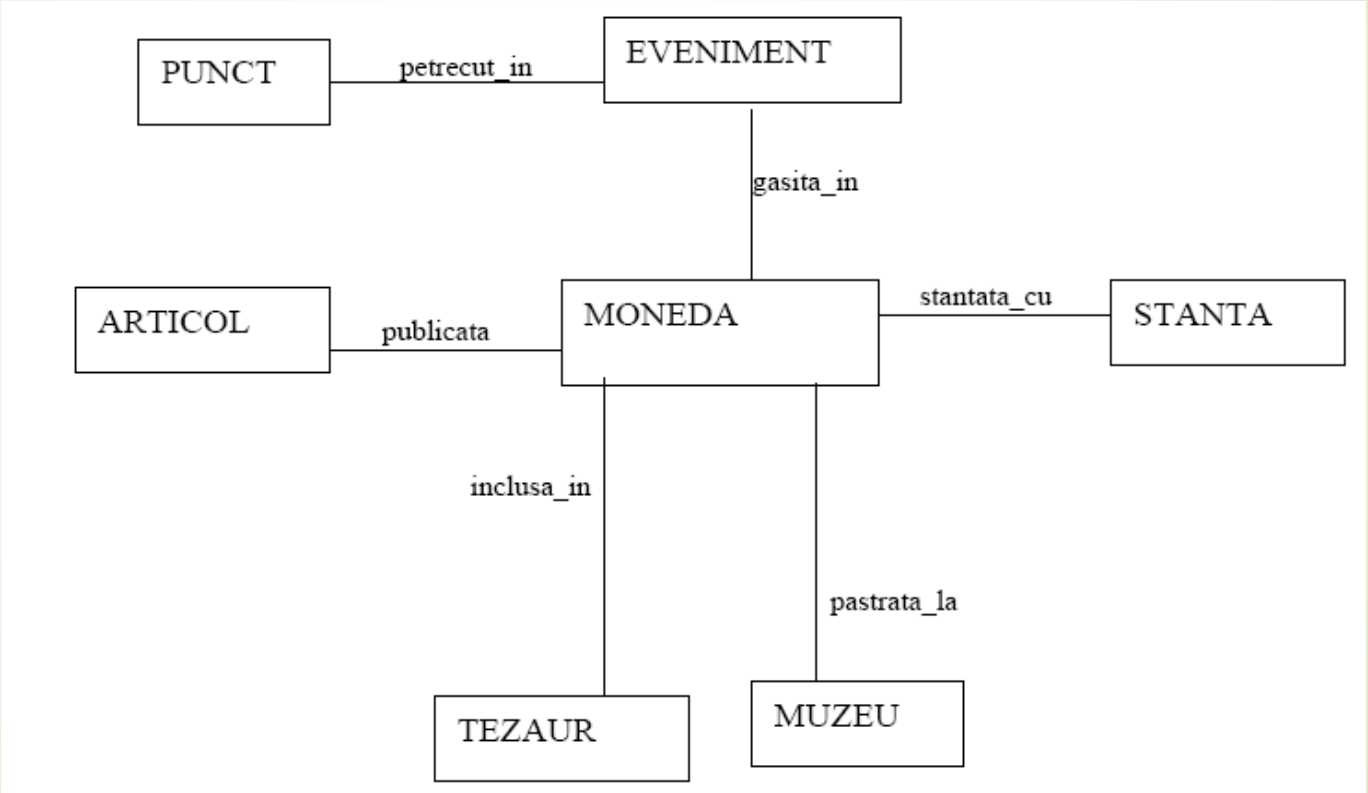
- Baza de date construită prin acest model, furnizează informații legate de obiective de execuție, investitori, executanți, şantiere, contracte etc. necesare unui manager al unei firme de construcții



Tabelele din cursurile *Oracle Education*..Tabelele *emp*, *dept*, *salgrade* modelează gestiunea salariaților unei firme.

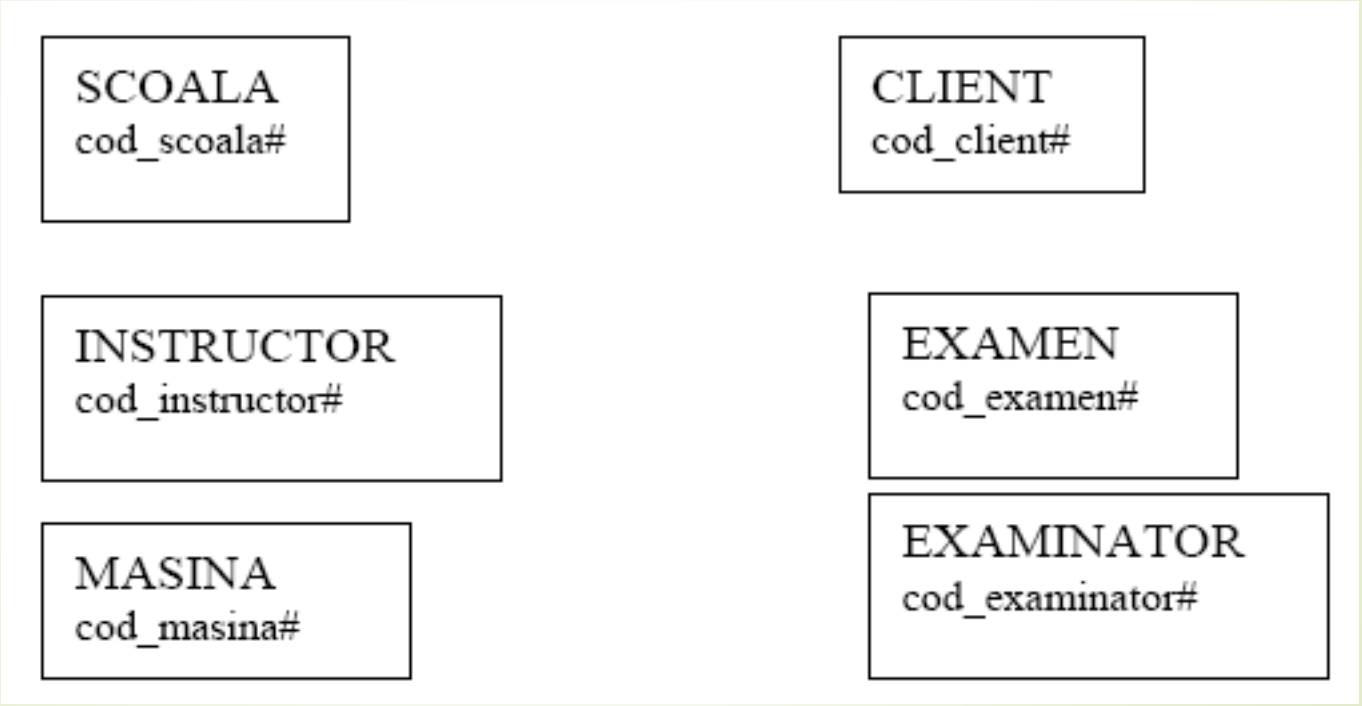
- Tabelul *emp*(empno#, ename, job, mgr, hiredate, sal, com, deptno) conține informații despre salariații unei firme. Pentru fiecare salariat sunt definite următoarele atribute: *empno*: codul salariatului; *ename*: numele salariatului; *job*: funcția; *mgr*: codul șefului; *hiredate*: data angajării; *sal*: salariul; *com*: comisionul; *deptno*: codul departamentului în care lucrează.
- Tabelul *dept* (deptno#, dname, loc) conține informații despre departamentele în care lucrează salariații. Atributele sale reprezintă: *deptno*: codul departamentului; *dname*: numele departamentului; loc: *localitatea* unde funcționează departamentul.
- Tabelul *salgrade*(grade#, losal, hisal) conține informații despre grilele de salarizare. Atributele tabelului au următoarea semnificație: *grade*: codul grilei de salarizare; *losal*: limita inferioară a grilei de salarizare; *hisal*: limita superioară a grilei de salarizare.

Ordonarea informațiilor cu privire la descoperirile de monede antice din Romania



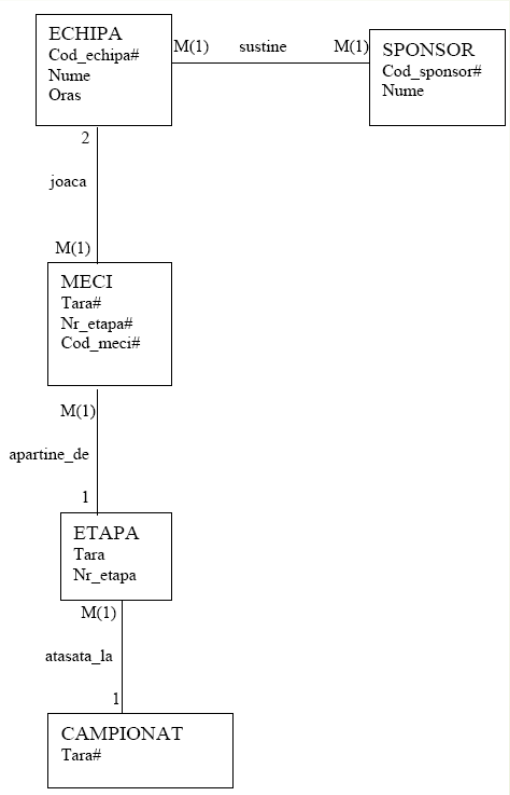
- STANȚA (nr_stanță, împărat emitent, valoare nominală, an emitere, monetăria, legenda de pe avers, legenda de pe revers) == > atribute ale entității **STANTA**
- Completați cardinalitatea!

Evidența şcolilor de şoferi din Romania



Completați relațiile (*lucreaza_la, conduce, sustine, asista, instruieste*) dintre entități şi specificați cardinalitatea!

Campionatele de fotbal ale diferitelor țări



Modelul relațional

- Modelul relațional a fost conceput şi dezvoltat de E.F. **Codd**. El este un model formal de organizare conceptuală a datelor, destinat reprezentării legăturilor dintre date, bazat pe teoria matematică a relațiilor. Modelul relațional este alcătuit numai din relații şi prin urmare, orice interogare asupra bazei de date este tot o relație. Cercetarea în domeniu → 3 mari proiecte (*System R*, *INGRES*, *PRTV*)
- **Calități:**
  - este simplu;
  - riguros din punct de vedere matematic;
  - nu este orientat spre sistemul de calcul.

Structura datelor

- Definirea noțiunilor de domeniu, relație, schemă relațională, valoare *null* şi tabel vizualizare (*view*).
- Conceptele utilizate pentru a descrie formal, uzual sau fizic elementele de ate în

Formal	Uzual	Fizic
relație	tablou	fişier
tuplu	linie	înregistrare
atribut	coloană	câmp
domeniu	tip de dată	tip de dată

- Domeniu – multime de valori care poate fi definită fie enumerând elementele componente, fie definind o proprietate distinctivă a domeniului valorilor.
- Fie $D1, D2, ..., Dn$ domenii finite, nu neapărat disjuncte. Produsul cartezian $D1 \times D2 \times ... \times Dn$ al domeniilor $D1, D2, ..., Dn$ este definit de multimea tuplurilor $(V1, V2, ..., Vn)$, unde $V1 \in D1, V2 \in D2, ..., Vn \in Dn$. Numărul n defineşte aritatea tuplului.

- O **relație** *R* pe mulțimile *D*1, *D*2, ..., *Dn* este o submulțime a produsului cartezian *D*1 × *D*2 × ... × *Dn*, deci este o mulțime de tupluri.
- Caracteristicile unei relatii:
  - are o denumire unica;
  - fiecare celula contine o valoare atomica;
  - fiecare atribut are nume unic;
  - toate valorile unui atribut apartin aceluiasi domeniu;
  - ordinea atributelor nu are importanta;
  - nu exista dubluri ale tuplurilor;
  - teoretic, ordinea tuplurilor nu are importanta.

- Definirea unei relații se referă la mulțimi care variază în timp. Pentru a caracteriza o relație este necesară existența un element invariant în timp: structura relației (schema relațională). Mulțimea numelor atributelor corespunzătoare unei relații *R* definește **schema relațională** a relației respective. Vom nota schema relațională prin *R*(*A*1, *A*2, ..., *An*). Exemplu!
- Putem reprezenta o relație printr-un tabel bidimensional în care fiecare linie corespunde unui tuplu și fiecare coloană corespunde unui domeniu din produsul cartezian. O coloană corespunde de fapt unui atribut. Numărul atributelor definește **gradul** (aritatea) relației, iar numărul de tupluri din relație definește **cardinalitatea** relației.

- Exemplu (crearea unui tabel în *SQL*):

```
CREATE TABLE salariat (
    cod_salariat        SMALLINT,
    nume                VARCHAR(25),
    prenume             VARCHAR(20),
    salariu             INTEGER,
    sot                 SMALLINT,
    job_cod             VARCHAR(6),
    cod_departament     SMALLINT );
```

- Când se inserează tupluri într-o relație, de multe ori un atribut este necunoscut sau neaplicabil. Pentru a reprezenta acest atribut a fost introdusă o valoare convențională în relație, şi anume valoarea ***null***.
- Este necesară o aritmetică şi o logică nouă care să cuprindă acest element. Rezultatul operatorilor aritmetici sau logici este *null* când unul din argumente este *null*. Comentat excepții! Prin urmare, „*null* = *null*“ are valoarea *null*, iar ¬ *null* este *null*.

AND	T	F	Null
T	T	F	Null
F	F	F	F
Null	Null	F	Null

OR	T	F	Null
T	T	T	T
F	T	F	Null
Null	T	Null	Null

Tabele de adevăr pentru operatorii AND şi OR.

- Multe din echivalentele adevarate in logica cu 2 valori nu mai sunt adevarate in aceasta logica (3VL).
  - De exemplu comparatia x = x nu are in mod necesar valoarea *true*; expresia *p OR NOT (p)* nu are in mod necesar valoarea *true*; expresia *t JOIN t* nu este evaluata neaparat ca fiind egala cu *t*, deoarece *join*-ul, spre deosebire de reuniune, se bazeaza pe verificarea egalitatii in „stil de gasire", nu in „stil de duplicat".
- Se observa ca *null*-urile ruineaza modelul relational. Logica 3VL nu corespunde realitatii, adica rezultatele care sunt corecte conform acestei logici sunt uneori eronate in lumea reala.
- Modelul relational a functionat perfect fara *null* in perioada 1969-1979.
- Este preferabil (in multe cazuri) utilizarea unor valori speciale pentru a reprezenta informatiile lipsa. De exemplu, se poate utiliza valoarea speciala „?" pentru a indica numarul de ore lucrate de un salariat. Atentie! Valoarea speciala sa fie de tipul aplicabil. In *SQL*, tratarea informatiilor lipsa se bazeaza substantial pe logica 3VL.

- Tabelul **vizualizare** (*view*, filtru, relație virtuală, vedere) constituie un filtru relativ la unul sau mai multe tabele, care conține numai informația necesară unei anumite abordări sau aplicații. Consultarea vizualizarilor functioneaza perfect.
- Vizualizarea este virtuală deoarece datele pe care le conține nu sunt în realitate memorate într-o bază de date. Este memorată numai definiția vizualizării. Vizualizarea nu este definită explicit, ca relațiile de bază, prin mulțimea tuplurilor componente, ci implicit, pe baza altor relații prin intermediul unor expresii relaționale. Stabilirea efectivă a tuplurilor care compun vizualizarea se realizează prin evaluarea expresiei atunci când utilizatorul se referă la acest tabel.

- Exemplu (crearea unei vizualizări în *SQL*):

```
CREATE VIEW
   programator(nume,departament)
 AS SELECT  nume,cod_departament
    FROM    salariat
    WHERE   job_cod='programator';
```

- Reguli de integritate → asertiuni pe care datele conținute în baza de date trebuie să le satisfacă.
- Trebuie făcută distincția între:
  - regulile structurale inerente modelării datelor;
  - regulile de funcționare specifice unei aplicații particulare.
- Există trei tipuri de constrângeri structurale (de cheie, de referință, de entitate) ce constituie mulțimea minimală de reguli de integritate pe care trebuie să le respecte un SGBD relațional. Restricțiile de integritate minimale sunt definite în raport cu noțiunea de cheie a unei relații.
- O mulțime minimală de atribute ale căror valori identifică unic un tuplu într-o relație reprezintă o cheie pentru relația respectivă.

- Fiecare relație are cel puțin o cheie. Una dintre cheile candidat va fi aleasă pentru a identifica efectiv tupluri și ea va primi numele de **cheie primară**. Cheia primară nu poate fi reactualizată. Atributele care reprezintă cheia primară sunt fie subliniate, fie urmate de semnul #.
- O cheie identifică linii și este diferită de un index care localizează liniile. O **cheie secundară** este folosită ca index pentru a accesa tupluri. Un grup de atribute din cadrul unei relații care conține o cheie a relației poartă numele de **supercheie**.

- Fie schemele relaționale *R*1(*P*1, *S*1) și *R*2(*S*1, *S*2), unde *P*1 este cheie primară pentru *R*1, *S*1 este cheie secundară pentru *R*1, iar *S*1 este cheie primară pentru *R*2. În acest caz, vom spune că *S*1 este **cheie externă** (cheie străină) pentru *R*1.

- Modelul relațional respectă trei reguli de integritate structurală.
- **Regula 1** – unicitatea cheii. Cheia primară trebuie să fie unică şi minimală.
- **Regula 2** – integritatea entității. Atributele cheii primare trebuie să fie diferite de valoarea *null*.
- **Regula 3 –** integritatea referirii. O cheie externă trebuie să fie ori *null* în întregime, ori să corespundă unei valori a cheii primare asociate.

Proiectarea modelului relațional

Transformarea entităților

- Entitățile independente devin **tabele independente**. Cheia primară nu conține chei externe.
- Entitățile dependente devin **tabele dependente**. Cheia primară a entităților dependente conține cheia primară a entității de care depinde (cheie externă) plus unul sau mai multe atribute adiționale.
- Subentitățile devin **subtabele**. Cheia externă se referă la supertabel, iar cheia primară este această cheie externă (cheia primară a subentității PROGRAMATOR este *cod_salariat* care este o cheie externă).

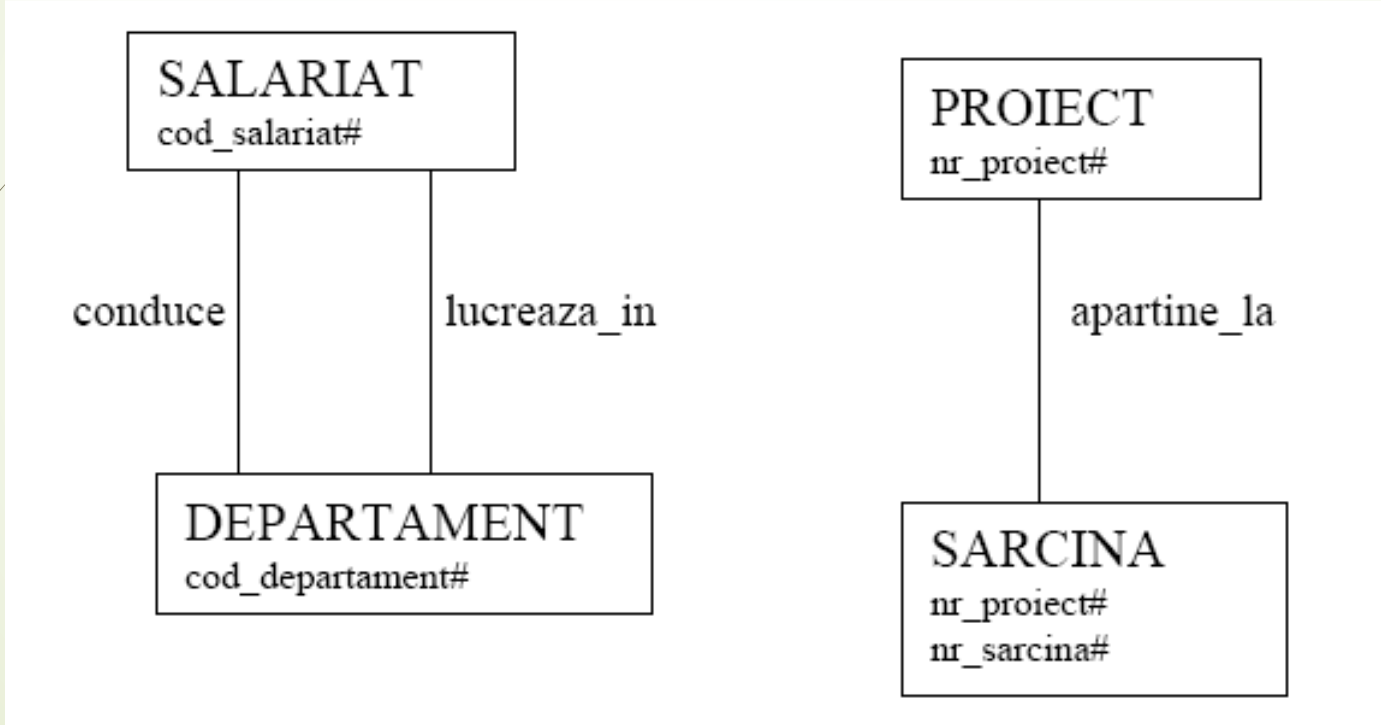
Transformarea relațiilor

- Relațiile 1:1 şi 1:*n* devin chei externe. Relația *conduce* devine coloană în tabelul DEPARTAMENT, iar relația *lucreaza_in* devine coloană în tabelul SALARIAT.
  - Simbolul „×“ indică plasamentul cheii externe, iar simbolul „$\underline{\times}$“ exprimă faptul că această cheie externă este conținută în cheia primară. Relația 1:1 plasează cheia externă în tabelul cu mai puține linii.
- Relația *m*:*n* devine un tabel special, numit tabel asociativ, care are două chei externe pentru cele două tabele asociate. Cheia primară este compunerea acestor două chei externe plus eventuale coloane adiționale. Tabelul se desenează punctat.
- Relațiile de tip trei devin tabele asociative. Cheia primară este compunerea a trei chei externe plus eventuale coloane adiționale.

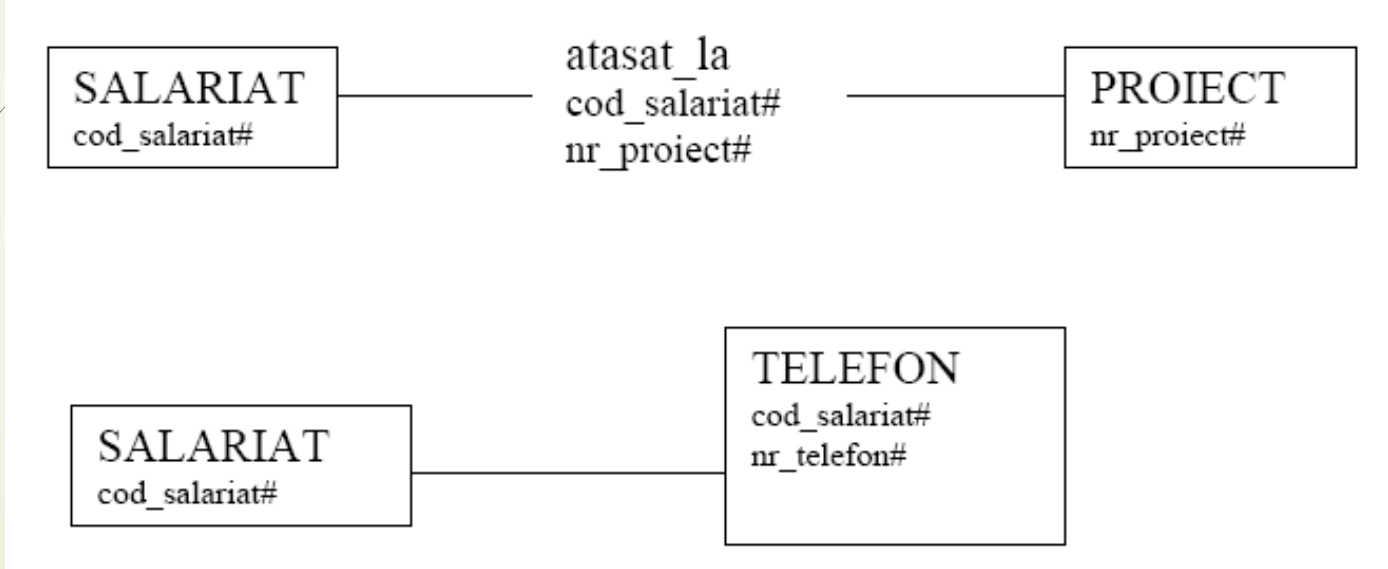
Transformarea atributelor

- Un atribut singular devine o coloană.
- Atributele multiple devin tabele dependendente ce conțin cheia primară a entității şi atributul multiplu. Cheia primară este o cheie externă, plus una sau mai multe coloane adiționale.
- Entitățile devin tabele, iar atributele lor devin coloane în aceste tabele.
- Ce devin atributele relațiilor? Pentru relații 1:1 şi 1:*n*, atributele relațiilor vor aparține tabelului care conține cheia externă, iar pentru relații *m*:*n* şi de tipul trei, atributele vor fi plasate în tabelele asociative.

Exemple (1)


SALARIAT
cod_salariat#
conduce
lucreaza_in
DEPARTAMENT
cod_departament#
PROIECT
nr_proiect#
apartine_la
SARCINA
nr_proiect#
nr_sarcina#

Exemple (2)

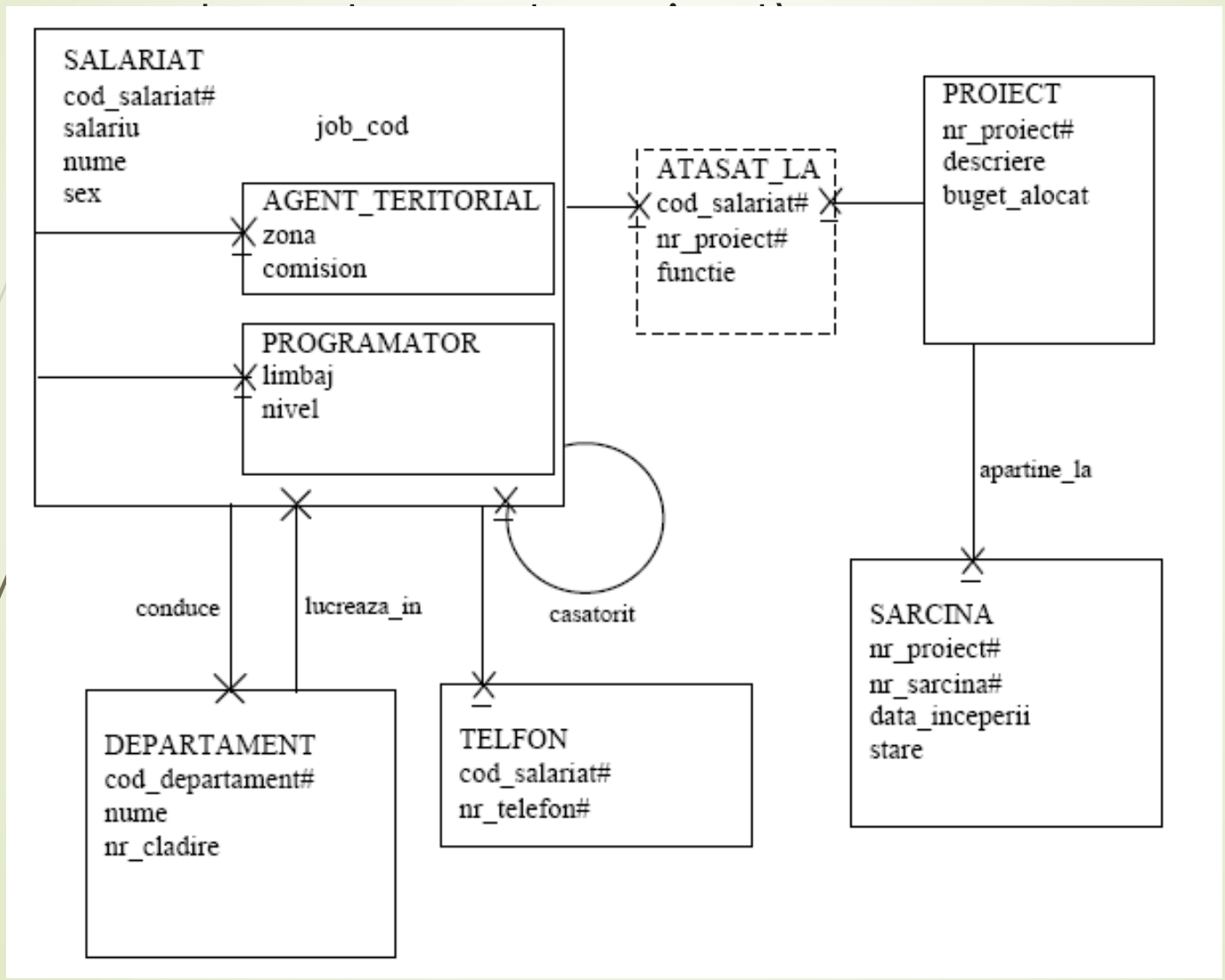

SALARIAT
cod_salariat#
atasat_la
cod_salariat#
nr_proiect#
PROIECT
nr_proiect#
SALARIAT
cod_salariat#
TELEFON
cod_salariat#
nr_telefon#

- Cele patru tipuri de tabele (independente, dependente, subtabele

Tabel	Reprezintă	Cheie primară
Independent	entitate independentă	nu conține chei externe
Subtabel	subentitate	o cheie externă
Dependent	entitate dependentă	o cheie externă şi una sau mai multe coloane adiționale
	atribute multiple	
Asociativ	relație m:n	două sau mai multe chei externe şi (opțional) coloane adiționale
	relații de tip 3	

- **Diagrama conceptuală pentru proiectarea modelului relaţional comentat a fost construită din diagrama E/R prin adăugarea tabelelor asociative şi prin marcarea cheilor externe.**

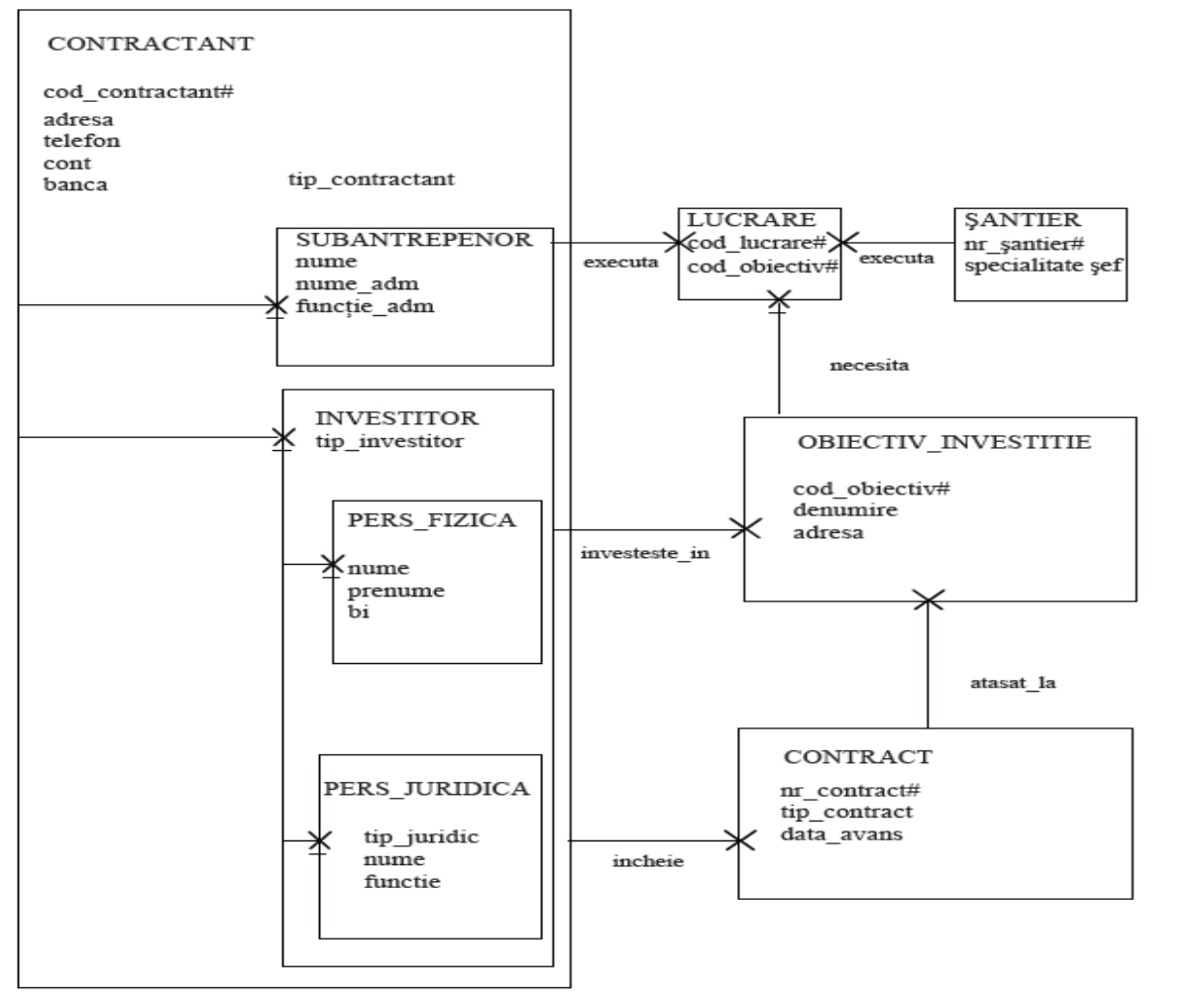
Exemplu diagramă conceptuală (salariat,



Schemele relaționale corespunzătoare acestei diagrame conceptuale sunt următoarele:

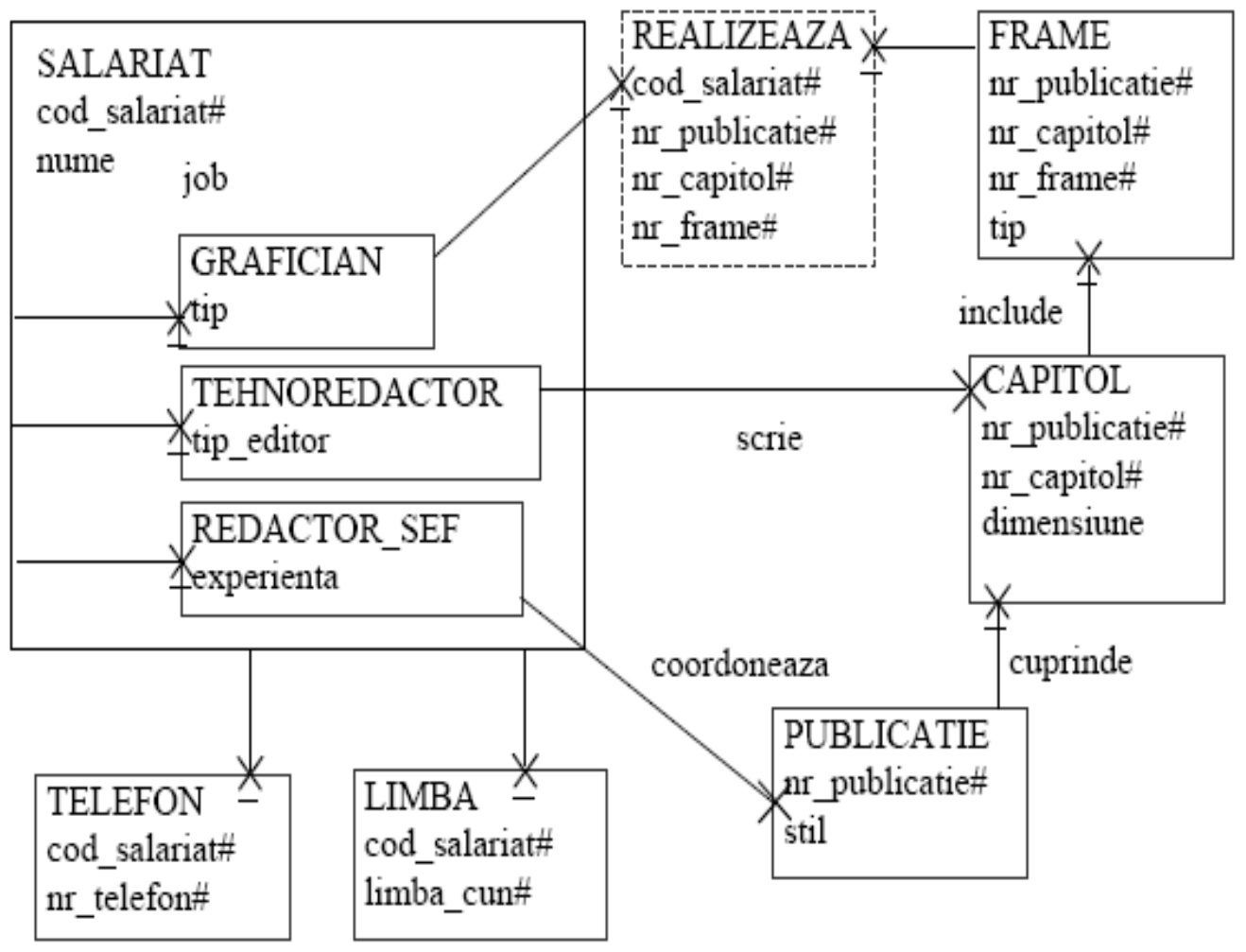
- SALARIAT(cod_salariat#, nume, prenume, sex, job_cod, cod_sot, forma_plata, nr_depart);
- DEPARTAMENT(cod_departament#, nume, numar_cladire, cod_sal);
- ATASAT_LA(cod_salariat#, nr_proiect#, functia);
- PROIECT(nr_proiect#, descriere, buget_alocat);
- SARCINA(nr_proiect#, nr_sarcina, data_inceperii, stare);
- AGENT_TERITORIAL(cod_salariat#, zona, comision);
- PROGRAMATOR(cod_salariat#, limbaj, nivel);
- TELEFON(cod_salariat#, nr_telefon#).

Exemplu (gestiunea activităților unei firme de construcții)



- CONTRACTANT(cod_contractant#, adresa, telefon, cont, banca, tip_contractant);
- SUBANTREPRENOR(cod_contractant#, nume, nr_reg_comert, nume_adm, functie_adm);
- INVESTITOR(cod_contractant#, tip_investitor);
- PERS_FIZICA(cod_contractant#, nume, prenume, bi);
- PERS_JURIDICA(cod_contractant#, tip_juridic, nume, reprez_legal, functie);
- CONTRACT(nr_contract#, tip_contract, data_incheiere, garantie, val_investitie, durate_executie, cont, banca, perioada, avans, data_avans, cod_contractant);
- SANTIER(nr_santier#, specialitate, sef);
- OBIECTIV_INVESTITIE(cod_obiectiv#, denumire, adresa, adc, nr_cert_urb, nr_aut_constr, nr_contract, cod_contractant);
- LUCRARE(cod_lucrare#, cod_obiectiv#, tip_lucrare, nume, data_inc, data_sf, nr_santier, cod_contractant);

Exemplu (Gestiunea activităților de editare dintr-o editură)



- SALARIAT(cod_salariat#, nume, prenume, vechime, salariu, job);
- GRAFICIAN(cod_salariat#, tip);
- TEHNOREDACTOR(cod_salariat#, tip_platforma, tip_editor, viteza);
- REDACTOR_SEF(cod_salariat#, experienta);
- LIMBA(cod_salariat#, limba_cunos#);
- TELEFON(cod_salariat#, nr_telefon#);
- REALIZEAZA(cod_salariat#, nr_frame#, nr_publicatie#, nr_capitol#, data_inc, data_lim);
- FRAME(nr_frame#, nr_publicatie#, nr_capitol#, tip, dim, format);
- CAPITOL(nr_publicatie#, nr_capitol#, dimensiune, cod_salariat);
- PUBLICATIE(nr_publicatie#, stil, beneficiar, autor, cod_salariat, cost, titlu, limba).

Algebra relațională

- Limbajul de definire a datelor (LDD) precizează entitățile, relațiile 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 relaționale cuprinde aspecte referitoare la introducerea, eliminarea, modificarea şi căutarea datelor.
  - **Introducerea** datelor – permite adăugarea de tupluri la o relație. Tuplurile pot fi introduse de utilizator sau pot fi obținute din alte relații existente în baza de date.
  - **Eliminarea** datelor – permite ştergerea tuplurilor ce satisfac condiții date.
  - **Modificarea** datelor – permite reactualizarea tuplurilor ce satisfac condiții date cu noi valori ale atributelor sau cu rezultate ale unor operații aritmetice efectuate asupra unor valori existente.
  - **Căutarea** datelor – permite găsirea tuplurilor sau a unor părți ale tuplurilor ce satisfac condiții date.

- Modelul relațional oferă două mulțimi de operatori pe relații:
  - algebra relațională (filtrele se obțin aplicând operatori specializați asupra uneia sau mai multor relații din cadrul bazei relaționale);
  - calculul relațional (filtrele se obțin cu ajutorul unor formule logice pe care tuplurile rezultatului trebuie să le satisfacă).

- **Algebra relațională** a fost introdusă de E.F. Codd ca o mulțime de operații formale acționând asupra unor relații şi având ca rezultat alte relații.
- Baza teoretică pentru limbajele de interogare relaționale o constituie operatorii introduşi de Codd pentru prelucrarea relațiilor.
- Operatorii modelului relațional definesc operațiile care se pot efectua asupra relațiilor în scopul realizării funcțiilor 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 intențiilor utilizatorului şi pot fi supuse unei diversități de reguli de transformare (optimizare).

- Relațiile sunt închise față de algebra relațională (operanzii şi rezultatele sunt relații → ieşirea unei operații poate deveni intrare pentru alta) → posibilitatea imbricării rxpresiilor în algebra relațională).
- Operatorii algebrei relaționale sunt:
  - operatori tradiționali pe mulțimi (UNION, INTERSECT, PRODUCT, DIFFERENCE);
  - operatori relaționali speciali (PROJECT, SELECT, JOIN, DIVISION).

- **Calculul relațional** reprezintă o adaptare a calculului predicatelor la domeniul bazelor de date relaționale. Ideea de bază este de a identifica o relație cu un predicat. Pe baza unor predicate inițiale, prin aplicarea unor operatori ai calculului cu predicate (conjuncția, disjuncția, negația, cuantificatorul existențial şi cel universal) se pot defini noi relații.
- Calculul relațional poate să fie orientat pe tupluri sau orientat pe domenii.
- Echivalența dintre algebra relațională şi calculul relațional a fost demonstrată de J.D.Ullman. Această echivalență arată că orice relație posibil de definit în algebra relațională poate fi definită şi în cadrul calculului relațional, şi reciproc.

Operatorii (unari sau binari) algebrei relaționale (1)

- SELECT (selecție) – extrage tupluri ce satisfac o condiție specificată;
- PROJECT (proiecție) – extrage atributele specificate;
- DIFFERENCE (diferență) – extrage tupluri care apar într-o relație, dar nu apar în cealaltă;
- PRODUCT (produs cartezian) – generează toate perechile posibile de tupluri, primul element al perechii fiind luat din prima relație, iar cel de-al doilea element din cealaltă relație;
- UNION (reuniune) – reuneşte două relații;
- INTERSECT (intersecție) – extrage tupluri care apar în ambele relații;
- DIVISION (diviziune) – extrage valorile atributelor dintr-o relație, care apar în toate valorile atributelor din cealaltă relație;

Operatorii (unari sau binari) algebrei relaționale (2)

- JOIN (compunere) – extrage tupluri din mai multe relații corelate:
- NATURAL JOIN (compunere naturală) – combină tupluri din două relații, cu condiția ca atributele comune să aibă valori identice;
- SEMI-JOIN (semi-compunere) – selectează tupluri ce aparțin unei singure relații, care sunt corelate cu tupluri din cea de a doua relație;
- Θ-JOIN (Θ-compunere) – combină tupluri din două relații (nu neaparat corelate), cu condiția ca valorile atributelor specificate să satisfacă o anumită condiție;
- OUTER JOIN (compunere externă) – combină tupluri din două relații, astfel încât condițiile de corelare să fie satisfăcute. Tuplurile din orice relație care nu satisfac aceste condiții sunt completate cu valori *null*.
- Pentru operatorii UNION, INTERSECT şi DIFFERENCE, se presupune că sunt aplicați numai la relații având aceeaşi aritate, iar ordinea (nu numele) atributelor este aceeaşi.

Operatorul PROJECT (1)

- Proiecția este o operație unară care elimină anumite atribute ale unei relații producând o submulțime „pe verticală“ a acesteia. Suprimarea unor atribute poate avea ca efect apariția unor tupluri duplicate, care trebuie eliminate.
- Prin proiecție se construieşte dintr-o relație *R*, o nouă relație:
  - ştergând din *R* atributele care nu sunt menționate în parametrii proiecției;
  - eliminând dublurile care apar după ştergere.
- Pentru a reprezenta operatorul proiecție sunt utilizate diferite **notații**:
  - $\Pi_{A1, \ldots, Am}$ (*R*)
  - PROJECT (*R*, *A*1, ..., *Am*)
  - *R*[*A*1, ..., *Am*]

  unde *A*1, *A*2, ..., *Am* sunt parametrii proiecției relativ la relația *R*.

Operatorul PROJECT (2)

***Exemplu*.** Să se obțină o listă ce conține numele, prenumele şi sexul angajaților.

- Proiecție în algebra relațională:
  - Rezultat = PROJECT(SALARIAT, nume, prenume, sex)
- Proiecție cu dubluri în SQL:

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

- Proiecție fără dubluri în SQL:

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

Operatorul SELECT

- Selecția (restrictia) este o operație unară care produce o submulțime pe „orizontală" a unei relații *R*. Această submulțime se obține prin extragerea tuplurilor din *R* care satisfac o condiție specificată.
- Sunt utilizate diferite notații:
  - $\sigma_{\text{condiție}}(R)$
  - *R*[condiție]
  - SELECT(*R*, condiție)
  - RESTRICT(*R*, condiție).
- ***Exemplu***. Să se obțină informații complete despre angajații din Bucureşti.
- Selecție în algebra relațională:

  Rezultat = SELECT(SALARIAT, oras = 'Bucuresti')
- Selecție în SQL:

```
SELECT   *
FROM     salariat
WHERE    oras = 'Bucuresti';
```

Operatorul UNION

- Reuniunea a două relații *R* şi *S* este mulțimea tuplurilor aparținând fie lui *R*, fie lui *S*, fie ambelor relații.
- Sunt utilizate notațiile:
  - $R \cup S$
  - UNION(*R*, *S*)
  - OR(*R*, *S*)
  - APPEND(*R*, *S*).
- ***Exemplu*.** Să se obțină lista cu numele persoanelor fizice şi a subantreprenorilor.

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

Operatorul DIFFERENCE

- Diferența a două relații *R* și *S* este mulțimea tuplurilor care aparțin lui *R*, dar nu aparțin lui *S*. Diferența este o operație binară necomutativă care permite obținerea tuplurilor ce apar numai într-o relație.
- Sunt utilizate diferite notații:
  - *R* – *S*
  - DIFFERENCE(*R*, *S*)
  - REMOVE(*R*, *S*)
  - MINUS(*R*, *S*).

***Exemplu*.** Să se obțină lista cu numărul contractului, tipul contractului, valoarea investiției și durata lucrării pentru contractele de subantrepriză pentru care valoarea investiției nu depăşeşte 60000$.

- Diferență în algebra relațională:

  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*)

- Diferența 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 diferența se poate referi la tabele diferite! Implementați cererea prin care se listează toate oraşele în care se află o filială, dar nici o proprietate.

Operatorul INTERSECT

- Intersecția a două relații *R* şi *S* este mulțimea tuplurilor care aparțin 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 notații:
  - INTERSECT(*R*, *S*)
  - $R \cap S$
  - AND(*R*, *S*).
- În anumite dialecte SQL există operator special (INTERSECT), care realizează această operație. Operatorii INTERSECT şi DIFFERENCE pot fi simulați în SQL (în cadrul comenzii SELECT) cu ajutorul opțiunilor EXISTS, NOT EXISTS, IN, != ANY.
- ***Exemplu.*** Utilizând tabelele *agent_teritorial* şi *programator* să se obțină lista codurilor salariaților care sunt programatori, dar care lucrează şi ca agenți teritoriali.
- Intersecție în algebra relațională:

  R = PROJECT(AGENT_TERITORIAL, cod_salariat);

  S = PROJECT(PROGRAMATOR, cod_salariat),

  Rezultat = INTERSECT(*R*, *S*).

- Intersecție în SQL:

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

- Simularea intersecției 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* și *S* relații de aritate *m*, respectiv *n*. Produsul cartezian al lui *R* cu *S* este mulțimea 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 notații:
  - $R \times S$
  - PRODUCT(*R*, *S*)
  - TIMES(*R*, *S*).
- ***Exemplu*.** Să se obțină lista tuturor posibilităților de investiție în diverse obiective de către o firmă care este persoană juridică.
- Produs cartezian în algebra relațională:

  R = PROJECT(PERS_JURIDICA, nume, cod_contractant);

  S = PROJECT(OBIECTIV_INVESTITIE, denumire);

  Rezultat = PRODUCT(*R*, *S*).
- Produs cartezian în SQL:

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

Operatorul DIVISION (1)

- Diviziunea este o operație binară care defineşte o relație ce conține valorile atributelor dintr-o relație care apar **în toate** valorile atributelor din cealaltă relație.
- Sunt utilizate diferite notații:
  - DIVIDE(*R*, *S*)
  - DIVISION(*R*, *S*)
  - $R \div S$.
- Diviziunea conține acele tupluri de dimensiune $n - m$ la care, adăugând orice tuplu din *S*, se obține un tuplu din *R*.

Operatorul DIVISION (2)

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

- Operatorul DIVISION este legat de cuantificatorul universal (∀) care nu există în SQL. Cuantificatorul universal poate fi însă simulat cu ajutorul cuantificatorului existențial (∃) utilizând relația:

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

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

Operatorul DIVISION (3)

- ***Exemplu***. Să se obțină codurile salariaților ataşați tuturor proiectelor pentru care s-a alocat un buget egal cu 1000.
- Diviziune în algebra relațională:

  R = PROJECT(ATASAT_LA, cod_salariat, nr_proiect);

  S = PROJECT(SELECT(PROIECT, buget = 1000), nr_proiect);

  Rezultat = DIVISION(*R*, *S*).
- 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));
```

Operatorul DIVISION (4)

- Simularea diviziunii cu ajutorul funcției 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 informației din mai multe relații corelate. Operatorul combină produsul cartezian, selecția şi proiecția.

Operatorul NATURAL JOIN (1)

- Operatorul de compunere naturală (NATURAL JOIN) combină tupluri din două relații *R* şi *S*, cu condiția 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 (2)

- 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, ..., *Ak* sunt atributele comune lui *R* şi *S*, iar *i*1, ..., *im* reprezintă lista componentelor din $R \times S$ (păstrând ordinea inițială) din care au fost eliminate componentele *S*.*A*1, ..., *S*.*Ak*.

- ***Exemplu***. Să se obțină informații complete despre angajați şi departamentele în care lucrează.
- Operatorul de compunere naturală în algebra relațională:

  Rezultat = JOIN(SALARIAT, DEPARTAMENT).
- Operatorul de compunere naturală în SQL:

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

Operatorul θ-JOIN (1)

- Operatorul **θ**-JOIN combină tupluri din două relații (nu neapărat corelate) cu condiția ca valorile atributelor specificate să satisfacă o anumită condiție specificată explicit în cadrul operației.
- Operatorul **θ**-JOIN este un operator derivat, fiind o combinație de produs scalar şi selecție:

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

- ***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;
```

Operatorul θ-JOIN (2)

- ***Exemplu*.** Să se obțină informații despre contractanți (codul şi banca) şi obiectivele de investiție asociate acestora (denumire, număr certificat de urbanizare) cu condiția ca obiectivele să nu fie la aceeaşi adresă ca şi contractanții.
- Operatorul **θ**-JOIN în algebra relațională:

  R = PROJECT(CONTRACTANT, cod_contractant, banca);

  S = PROJECT(OBIECTIV_INVESTITIE, denumire, nr_cert_urb);

  Rezultat = JOIN(*R*, *S*, OBIECTIV_INVESTITIE.adresa <>
  CONTRACTANT.adresa).
- 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 (1)

- Operatorul SEMI-JOIN conservă atributele unei singure relații participante la compunere şi este utilizat când nu sunt necesare toate atributele compunerii. Operatorul este asimetric.
- Tupluri ale relației R care participă în compunerea (naturală sau θ-JOIN) dintre relațiile R şi S.
- SEMI-JOIN este un operator derivat, fiind o combinație de proiecție şi compunere naturală sau proiecție şi θ-JOIN:

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

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

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

Operatorul SEMI-JOIN (2)

- ***Exemplu*.** Să se obțină informații referitoare la persoanele fizice (nume, buletin) care investesc în obiective cu caracter recreativ.
- Operatorul SEMI-JOIN în algebra relațională:

  R = SELECT(OBIECTIV_INVESTITIE, denumire = 'cabana' OR denumire = 'casa de vacanta')

  S = JOIN(PERS_FIZICA, *R*)

  Rezultat = PROJECT(*S*, nume, buletin).
- 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 (1)

- Operația de compunere externă combină tupluri din două relații pentru care sunt satisfăcute condițiile de corelare. În cazul aplicării operatorului JOIN se pot pierde tupluri, atunci când există un tuplu în una din relații pentru care nu există nici un tuplu în cealaltă relație, astfel încât să fie satisfăcută relația de corelare.
- Operatorul elimină acest inconvenient prin atribuirea valorii *null* valorilor atributelor care există într-un tuplu din una dintre relațiile de intrare, dar care nu există şi în cea de-a doua relație.
- Practic, se realizează compunerea a două relații *R* şi *S* la care se adaugă tupluri din *R* şi *S*, care nu sunt conținute 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 relația S, sunt de asemenea incluse în relația rezultat.

Operatorul OUTER JOIN (2)

- ***Exemplu*.** Să se obțină informații referitoare la persoanele fizice care sunt investitori (chiar dacă nu au investit în obiective industriale) şi la obiectivele de investiție industriale (chiar şi cele care nu sunt construite de persoane fizice).

  R = SELECT(OBIECTIV_INVESTITIE, denumire = 'industrial')

  Rezultat = OUTERJOIN(PERS_FIZICA, *R*).
- Operatorii algebrei relaționale pot fi reprezentați grafic cu ajutorul unor simboluri speciale.

Evaluarea şi optimizarea interogărilor

Procesarea interogărilor (1)

- O **expresie** a algebrei relaționale este constituită din relații legate prin operații din algebra relațională.
- 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 aparițiilor sau în ordinea fixată prin paranteze.
- Rezultatul evaluării unei expresii este o relație derivată din relațiile menționate ca operanzi în cadrul expresiei.
- Două expresii sunt **echivalente**, dacă în urma evaluării lor se obține ca rezultat aceeaşi relație.

Procesarea interogărilor (2)

- **Exemple** referitoare la moduri echivalente de exprimare a unei cereri.
- **1**. Informații despre salariații care nu contribuie la machetarea nici unei publicații, dar au retribuția mai mare decât o valoare dată.
- **2**. Codul şi numele subantreprenorilor care au realizat lucrări specializate la obiective case de vacanță sau cabane.
- **3**. Codurile şi telefoanele investitorilor, valoarea si durata de execuție a investitiilor a caror valoare este între două limite specificate.
- **4**. Perioada de desfăşurare şi prețul ofertelor care încep după 1 ianuarie 2003 şi sunt:
  - sejururi la munte;
  - excursii în care autocarele sunt conduse de şoferi angajați după 1 mai 1987 şi supravegheate de ghizi ce cunosc limba engleză care au făcut specializare în Suedia.

Procesarea interogărilor (3)

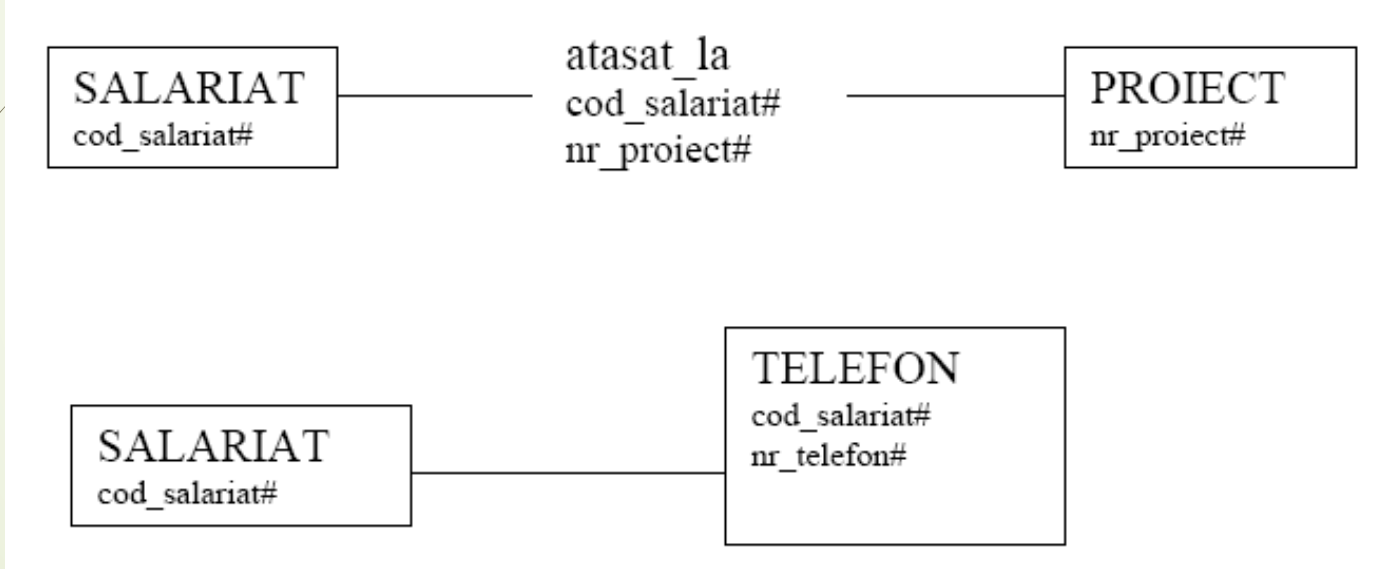
- În majoritatea sistemelor de gestiune, şi în special în cele relaționale, interfața cu utilizatorul este de tip **neprocedural**.
- Utilizatorul defineşte datele pe care doreşte să le vizualizeze fără a da algoritmii de acces. Sistemul trebuie să convertească cererea utilizatorului:
  - într-o cerere optimală;
  - în proceduri de acces optimal la datele fizice.
- Garantarea absolută a performanțelor optime pentru procesorul limbajului relațional este imposibilă. Corectă ar fi utilizarea cuvântului „ameliorare“ în locul cuvântului „optimizare“.

Procesarea interogărilor (4)

- **Evaluarea unei interogări** se efectuează în trei etape.

1. Analiza cererii presupune studierea sintactică şi semantică a cererii pentru a verifica corectitudinea sa şi a simplifica criteriul de căutare.
2. 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.
3. Execuţia presupune efectuarea (paralel şi/sau secvenţială) operaţiilor elementare furnizate de planul de execuţie pentru a obţine rezultatul cererii.

Exemple (2)

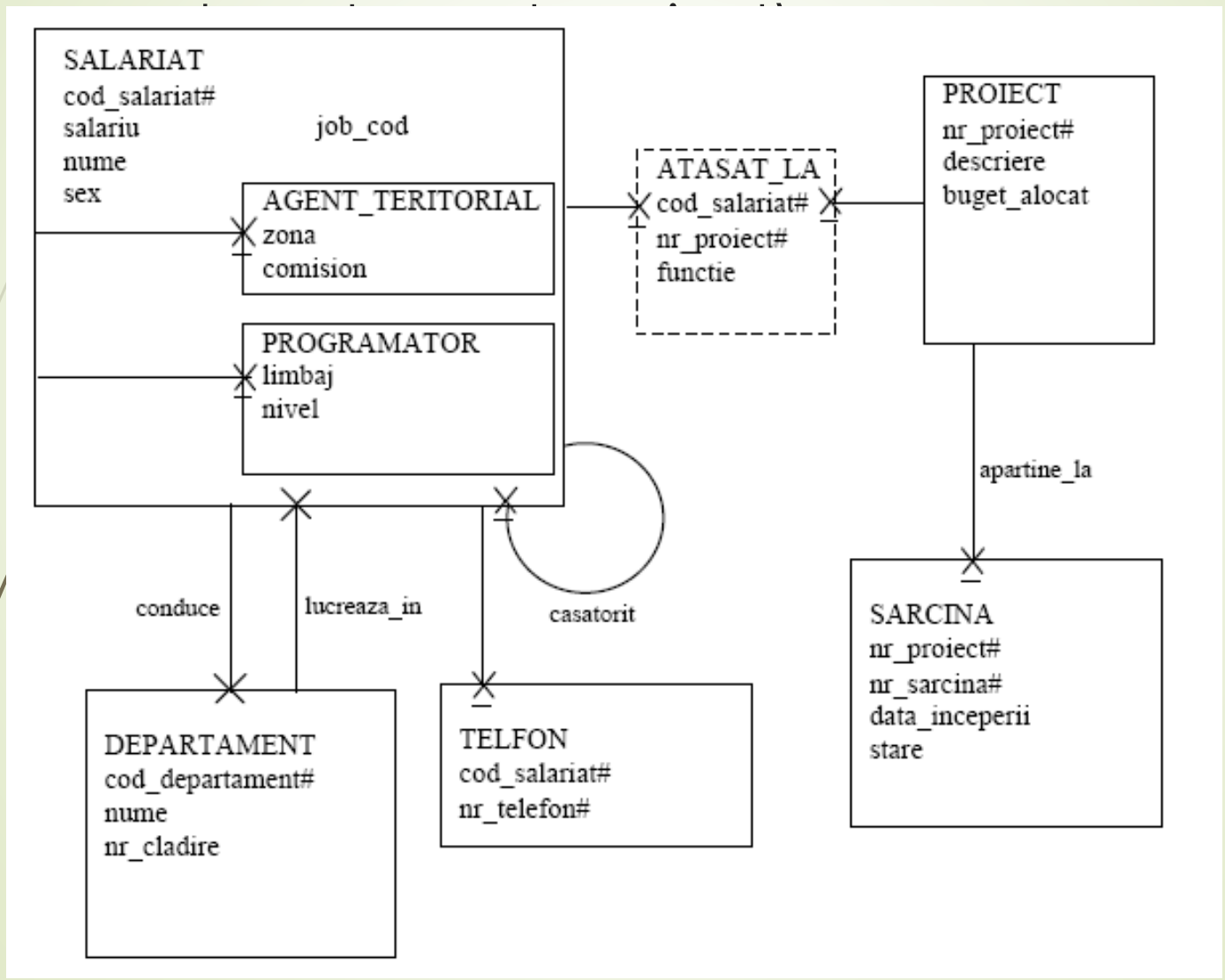

SALARIAT
cod_salariat#
atasat_la
cod_salariat#
nr_proiect#
PROIECT
nr_proiect#
SALARIAT
cod_salariat#
TELEFON
cod_salariat#
nr_telefon#

- Cele patru tipuri de tabele (independente, dependente, subtabele

Tabel	Reprezintă	Cheie primară
Independent	entitate independentă	nu conține chei externe
Subtabel	subentitate	o cheie externă
Dependent	entitate dependentă	o cheie externă şi una sau mai multe coloane adiționale
	atribute multiple	
Asociativ	relație m:n	două sau mai multe chei externe şi (opțional) coloane adiționale
	relații de tip 3	

- **Diagrama conceptuală pentru proiectarea modelului relaţional comentat a fost construită din diagrama E/R prin adăugarea tabelelor asociative şi prin marcarea cheilor externe.**

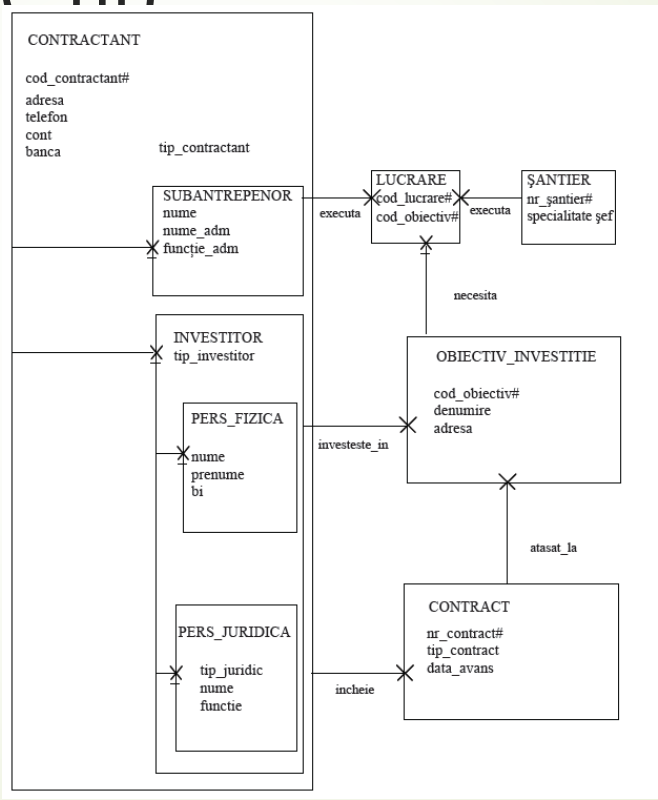
Exemplu diagramă conceptuală (salariat,



Schemele relaționale corespunzătoare acestei diagrame conceptuale sunt următoarele:

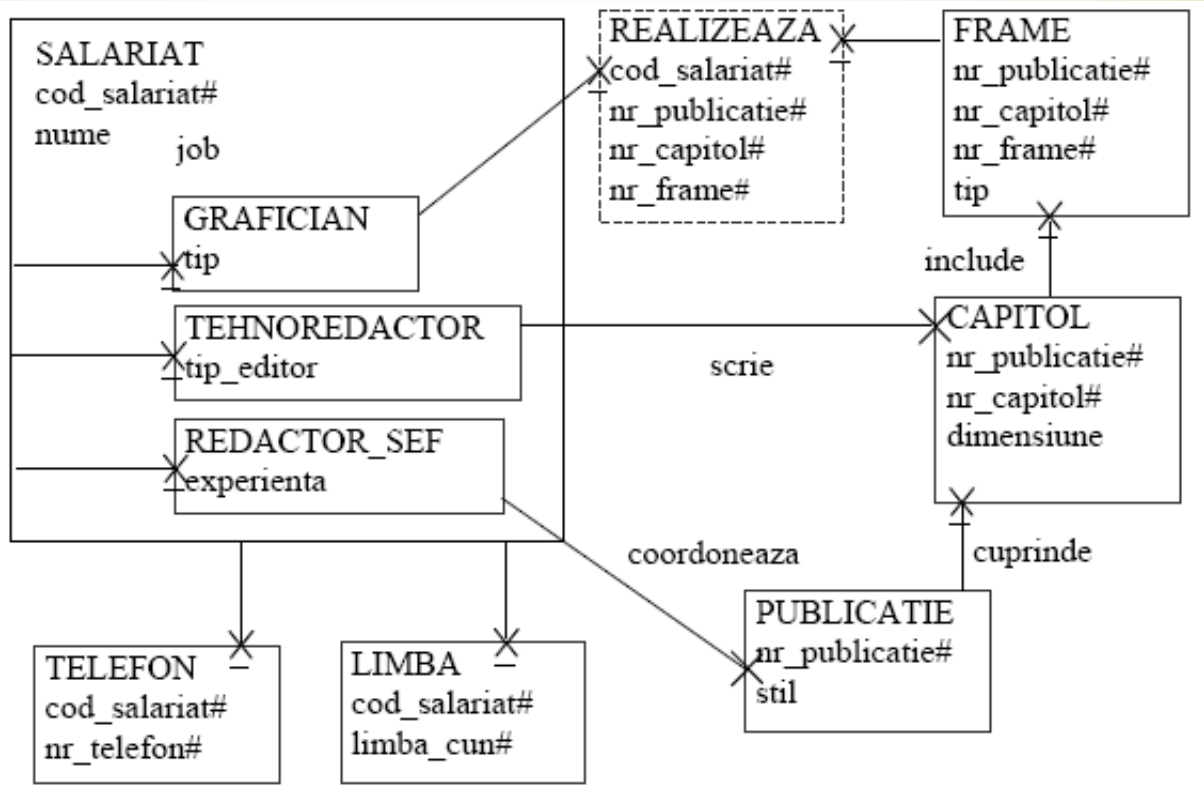
- SALARIAT(cod_salariat#, nume, prenume, sex, job_cod, cod_sot, forma_plata, nr_depart);
- DEPARTAMENT(cod_departament#, nume, numar_cladire, cod_sal);
- ATASAT_LA(cod_salariat#, nr_proiect#, functia);
- PROIECT(nr_proiect#, descriere, buget_alocat);
- SARCINA(nr_proiect#, nr_sarcina, data_inceperii, stare);
- AGENT_TERITORIAL(cod_salariat#, zona, comision);
- PROGRAMATOR(cod_salariat#, limbaj, nivel);
- TELEFON(cod_salariat#, nr_telefon#).

Exemplu (gestiunea activităților unei firme de construcții)



- CONTRACTANT(cod_contractant#, adresa, telefon, cont, banca, tip_contractant);
- SUBANTREPRENOR(cod_contractant#, nume, nr_reg_comert, nume_adm, functie_adm);
- INVESTITOR(cod_contractant#, tip_investitor);
- PERS_FIZICA(cod_contractant#, nume, prenume, bi);
- PERS_JURIDICA(cod_contractant#, tip_juridic, nume, reprez_legal, functie);
- CONTRACT(nr_contract#, tip_contract, data_incheiere, garantie, val_investitie, durate_executie, cont, banca, perioada, avans, data_avans, cod_contractant);
- SANTIER(nr_santier#, specialitate, sef);
- OBIECTIV_INVESTITIE(cod_obiectiv#, denumire, adresa, adc, nr_cert_urb, nr_aut_constr, nr_contract, cod_contractant);
- LUCRARE(cod_lucrare#, cod_obiectiv#, tip_lucrare, nume, data_inc, data_sf, nr_santier, cod_contractant);

Exemplu (Gestiunea activităților de editare dintr-o editură)



- SALARIAT(cod_salariat#, nume, prenume, vechime, salariu, job);
- GRAFICIAN(cod_salariat#, tip);
- TEHNOREDACTOR(cod_salariat#, tip_platforma, tip_editor, viteza);
- REDACTOR_SEF(cod_salariat#, experienta);
- LIMBA(cod_salariat#, limba_cunos#);
- TELEFON(cod_salariat#, nr_telefon#);
- REALIZEAZA(cod_salariat#, nr_frame#, nr_publicatie#, nr_capitol#, data_inc, data_lim);
- FRAME(nr_frame#, nr_publicatie#, nr_capitol#, tip, dim, format);
- CAPITOL(nr_publicatie#, nr_capitol#, dimensiune, cod_saldriat);
- PUBLICATIE(nr_publicatie#, stil, beneficiar, autor, cod_salariat, cost, titlu, limba).

Algebra relațională

- Limbajul de definire a datelor (LDD) precizează entitățile, relațiile 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 relaționale cuprinde aspecte referitoare la introducerea, eliminarea, modificarea şi căutarea datelor.
  - **Introducerea** datelor – permite adăugarea de tupluri la o relație. Tuplurile pot fi introduse de utilizator sau pot fi obținute din alte relații existente în baza de date.
  - **Eliminarea** datelor – permite ştergerea tuplurilor ce satisfac condiții date.
  - **Modificarea** datelor – permite reactualizarea tuplurilor ce satisfac condiții date cu noi valori ale atributelor sau cu rezultate ale unor operații aritmetice efectuate asupra unor valori existente.
  - **Căutarea** datelor – permite găsirea tuplurilor sau a unor părți ale tuplurilor ce satisfac condiții date.

- Modelul relațional oferă două mulțimi de operatori pe relații:
  - algebra relațională (filtrele se obțin aplicând operatori specializați asupra uneia sau mai multor relații din cadrul bazei relaționale);
  - calculul relațional (filtrele se obțin cu ajutorul unor formule logice pe care tuplurile rezultatului trebuie să le satisfacă).

- **Algebra relațională** a fost introdusă de E.F. Codd ca o mulțime de operații formale acționând asupra unor relații şi având ca rezultat alte relații.
- Baza teoretică pentru limbajele de interogare relaționale o constituie operatorii introduşi de Codd pentru prelucrarea relațiilor.
- Operatorii modelului relațional definesc operațiile care se pot efectua asupra relațiilor în scopul realizării funcțiilor 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 intențiilor utilizatorului şi pot fi supuse unei diversități de reguli de transformare (optimizare).

- Relațiile sunt închise față de algebra relațională (operanzii şi rezultatele sunt relații → ieşirea unei operații poate deveni intrare pentru alta) → posibilitatea imbricării expresiilor în algebra relațională).
- Operatorii algebrei relaționale sunt:
  - operatori tradiționali pe mulțimi (UNION, INTERSECT, PRODUCT, DIFFERENCE);
  - operatori relaționali speciali (PROJECT, SELECT, JOIN, DIVISION).

- **Calculul relațional** reprezintă o adaptare a calculului predicatelor la domeniul bazelor de date relaționale. Ideea de bază este de a identifica o relație cu un predicat. Pe baza unor predicate inițiale, prin aplicarea unor operatori ai calculului cu predicate (conjuncția, disjuncția, negația, cuantificatorul existențial şi cel universal) se pot defini noi relații.
- Calculul relațional poate să fie orientat pe tupluri sau orientat pe domenii.
- Echivalența dintre algebra relațională şi calculul relațional a fost demonstrată de J.D.Ullman. Această echivalență arată că orice relație posibil de definit în algebra relațională poate fi definită şi în cadrul calculului relațional, şi reciproc.

Operatorii (unari sau binari) algebrei relaționale (1)

- SELECT (selecție) – extrage tupluri ce satisfac o condiție specificată;
- PROJECT (proiecție) – extrage atributele specificate;
- DIFFERENCE (diferență) – extrage tupluri care apar într-o relație, dar nu apar în cealaltă;
- PRODUCT (produs cartezian) – generează toate perechile posibile de tupluri, primul element al perechii fiind luat din prima relație, iar cel de-al doilea element din cealaltă relație;
- UNION (reuniune) – reuneşte două relații;
- INTERSECT (intersecție) – extrage tupluri care apar în ambele relații;
- DIVISION (diviziune) – extrage valorile atributelor dintr-o relație, care apar în toate valorile atributelor din cealaltă relație;

Operatorii (unari sau binari) algebrei relaționale (2)

- JOIN (compunere) – extrage tupluri din mai multe relații corelate:
- NATURAL JOIN (compunere naturală) – combină tupluri din două relații, cu condiția ca atributele comune să aibă valori identice;
- SEMI-JOIN (semi-compunere) – selectează tupluri ce aparțin unei singure relații, care sunt corelate cu tupluri din cea de a doua relație;
- Θ-JOIN (Θ-compunere) – combină tupluri din două relații (nu neapărat corelate), cu condiția ca valorile atributelor specificate să satisfacă o anumită condiție;
- OUTER JOIN (compunere externă) – combină tupluri din două relații, astfel încât condițiile de corelare să fie satisfăcute. Tuplurile din orice relație care nu satisfac aceste condiții sunt completate cu valori *null*.
- Pentru operatorii UNION, INTERSECT şi DIFFERENCE, se presupune că sunt aplicați numai la relații având aceeaşi aritate, iar ordinea (nu numele) atributelor este aceeaşi.

Operatorul PROJECT (1)

- Proiecția este o operație unară care elimină anumite atribute ale unei relații producând o submulțime „pe verticală" a acesteia. Suprimarea unor atribute poate avea ca efect apariția unor tupluri duplicate, care trebuie eliminate.
- Prin proiecție se construieşte dintr-o relație *R*, o nouă relație:
  - ştergând din *R* atributele care nu sunt menționate în parametrii proiecției;
  - eliminând dublurile care apar după ştergere.
- Pentru a reprezenta operatorul proiecție sunt utilizate diferite **notații**:
  - $\Pi_{A1, ..., Am}$ (*R*)
  - PROJECT (*R*, *A*1, ..., *Am*)
  - *R*[*A*1, ..., *Am*]

  unde *A*1, *A*2, ..., *Am* sunt parametrii proiecției relativ la relația *R*.

Operatorul PROJECT (2)

Exemplu. Să se obțină o listă ce conține numele, prenumele şi sexul angajaților.

- Proiecție în algebra relațională:
  - Rezultat = PROJECT(SALARIAT, nume, prenume, sex)
- Proiecție cu dubluri în SQL:

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

- Proiecție fără dubluri în SQL:

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

Operatorul SELECT

- Selecția (restricția) este o operație unară care produce o submulțime pe „orizontală" a unei relații *R*. Această submulțime se obține prin extragerea tuplurilor din *R* care satisfac o condiție specificată.
- Sunt utilizate diferite notații:
  - $\sigma_{condiție}(R)$
  - *R*[condiție]
  - SELECT(*R*, condiție)
  - RESTRICT(*R*, condiție).
- ***Exemplu.*** Să se obțină informații complete despre angajații din Bucureşti.
- Selecție în algebra relațională:

  Rezultat = SELECT(SALARIAT, oras = 'Bucuresti')
- Selecție în SQL:

```
SELECT    *
FROM      salariat
WHERE     oras = 'Bucuresti';
```

Operatorul UNION

- Reuniunea a două relații *R* și *S* este mulțimea tuplurilor aparținând fie lui *R*, fie lui *S*, fie ambelor relații.
- Sunt utilizate notațiile:
  - $R \cup S$
  - UNION(*R*, *S*)
  - OR(*R*, *S*)
  - APPEND(*R*, *S*).
- ***Exemplu.*** Să se obțină lista cu numele persoanelor fizice și a subantreprenorilor.

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

Operatorul DIFFERENCE

- Diferența a două relații *R* și *S* este mulțimea tuplurilor care aparțin lui *R*, dar nu aparțin lui *S*. Diferența este o operație binară necomutativă care permite obținerea tuplurilor ce apar numai într-o relație.
- Sunt utilizate diferite notații:
  - *R – S*
  - DIFFERENCE(*R*, *S*)
  - REMOVE(*R*, *S*)
  - MINUS(*R*, *S*).

Exemplu. Să se obțină lista cu numărul contractului, tipul contractului, valoarea investiției și durata lucrării pentru contractele de subantrepriză pentru care valoarea investiției nu depaşeşte 60000$.

- Diferență în algebra relațională:

  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*)

- Diferența 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 diferenta se poate referi la tabele diferite! Implementati cererea prin care se listează toate orașele în care se află o filială, dar nici o proprietate.

Operatorul INTERSECT

- Intersecția a două relații *R* și *S* este mulțimea tuplurilor care aparțin 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 notații:
  - INTERSECT(*R*, *S*)
  - $R \cap S$
  - AND(*R*, *S*).
- În anumite dialecte SQL există operator special (INTERSECT), care realizează această operație. Operatorii INTERSECT și DIFFERENCE pot fi simulați în SQL (în cadrul comenzii SELECT) cu ajutorul opțiunilor EXISTS, NOT EXISTS, IN, != ANY.
- ***Exemplu.*** Utilizând tabelele *agent_teritorial* şi *programator* să se obțină lista codurilor salariaților care sunt programatori, dar care lucrează şi ca agenți teritoriali.
- Intersecție în algebra relațională:

  R = PROJECT(AGENT_TERITORIAL, cod_salariat);

  S = PROJECT(PROGRAMATOR, cod_salariat),

  Rezultat = INTERSECT(*R*, *S*).

- Intersecție în SQL:

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

- Simularea intersecției 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* și *S* relații de aritate *m*, respectiv *n*. Produsul cartezian al lui *R* cu *S* este mulțimea 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 notații:
  - $R \times S$
  - PRODUCT(*R*, *S*)
  - TIMES(*R*, *S*).
- ***Exemplu.*** Să se obțină lista tuturor posibilităților de investiție în diverse obiective de către o firmă care este persoană juridică.
- Produs cartezian în algebra relațională:

  R = PROJECT(PERS_JURIDICA, nume, cod_contractant);

  S = PROJECT(OBIECTIV_INVESTITIE, denumire);

  Rezultat = PRODUCT(*R*, *S*).
- Produs cartezian în SQL:

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

Operatorul DIVISION (1)

- Diviziunea este o operație binară care defineşte o relație ce conține valorile atributelor dintr-o relație care apar **în toate** valorile atributelor din cealaltă relație.
- Sunt utilizate diferite notații:
  - DIVIDE(*R*, *S*)
  - DIVISION(*R*, *S*)
  - $R \div S$.
- Diviziunea conține acele tupluri de dimensiune $n - m$ la care, adăugând orice tuplu din *S*, se obține un tuplu din *R*.

Operatorul DIVISION (2)

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

- Operatorul DIVISION este legat de cuantificatorul universal (∀) care nu există în SQL. Cuantificatorul universal poate fi însă simulat cu ajutorul cuantificatorului existențial (∃) utilizând relația:

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

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

Operatorul DIVISION (3)

- ***Exemplu.*** Să se obțină codurile salariaților ataşați tuturor proiectelor pentru care s-a alocat un buget egal cu 1000.
- Diviziune în algebra relațională:

  R = PROJECT(ATASAT_LA, cod_salariat, nr_proiect);

  S = PROJECT(SELECT(PROIECT, buget = 1000), nr_proiect);

  Rezultat = DIVISION(*R*, *S*).
- 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));
```

Operatorul DIVISION (4)

- Simularea diviziunii cu ajutorul funcției 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 informației din mai multe relații corelate. Operatorul combină produsul cartezian, selecția şi proiecția.

Operatorul NATURAL JOIN (1)

- Operatorul de compunere naturală (NATURAL JOIN) combină tupluri din două relații *R* şi *S*, cu condiția 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 definește 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 (2)

- 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, ..., *Ak* sunt atributele comune lui *R* şi *S*, iar *i*1, ..., *im* reprezintă lista componentelor din $R \times S$ (păstrând ordinea inițială) din care au fost eliminate componentele *S*.*A*1, ..., *S*.*Ak*.

- ***Exemplu***. Să se obțină informații complete despre angajați şi departamentele în care lucrează.
- Operatorul de compunere naturală în algebra relațională:

  Rezultat = JOIN(SALARIAT, DEPARTAMENT).
- Operatorul de compunere naturală în SQL:

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

Operatorul θ-JOIN (1)

- Operatorul **θ**-JOIN combină tupluri din două relații (nu neapărat corelate) cu condiția ca valorile atributelor specificate să satisfacă o anumită condiție specificată explicit în cadrul operației.
- Operatorul **θ**-JOIN este un operator derivat, fiind o combinație de produs scalar și selecție:

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

- ***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;
```

Operatorul θ-JOIN (2)

- ***Exemplu.*** Să se obțină informații despre contractanți (codul şi banca) şi obiectivele de investiție asociate acestora (denumire, număr certificat de urbanizare) cu condiția ca obiectivele să nu fie la aceeaşi adresă ca şi contractanții.
- Operatorul **θ**-JOIN în algebra relațională:

  R = PROJECT(CONTRACTANT, cod_contractant, banca);

  S = PROJECT(OBIECTIV_INVESTITIE, denumire, nr_cert_urb);

  Rezultat = JOIN(*R*, *S*, OBIECTIV_INVESTITIE.adresa <>
  CONTRACTANT.adresa).
- 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 (1)

- Operatorul SEMI-JOIN conservă atributele unei singure relații participante la compunere şi este utilizat când nu sunt necesare toate atributele compunerii. Operatorul este asimetric.
- Tupluri ale relației R care participă în compunerea (naturală sau θ-JOIN) dintre relațiile R şi S.
- SEMI-JOIN este un operator derivat, fiind o combinație de proiecție şi compunere naturală sau proiecție şi θ-JOIN:

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

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

Operatorul SEMI-JOIN (2)

- ***Exemplu.*** Să se obțină informații referitoare la persoanele fizice (nume, buletin) care investesc în obiective cu caracter recreativ.
- Operatorul SEMI-JOIN în algebra relațională:

  R = SELECT(OBIECTIV_INVESTITIE, denumire = 'cabana' OR denumire = 'casa de vacanta')

  S = JOIN(PERS_FIZICA, *R*)

  Rezultat = PROJECT(*S*, nume, buletin).
- 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 (1)

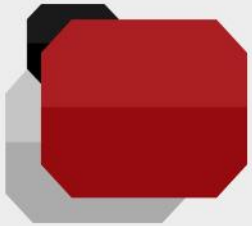
- Operaţia de compunere externă combină tupluri din două relaţii pentru care sunt satisfăcute condiţiile de corelare. În cazul aplicării operatorului JOIN se pot pierde tupluri, atunci când există un tuplu în una din relaţii pentru care nu există nici un tuplu în cealaltă relaţie, astfel încât să fie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

Operatorul OUTER JOIN (2)

- ***Exemplu.*** Să se obțină informații referitoare la persoanele fizice care sunt investitori (chiar dacă nu au investit în obiective industriale) şi la obiectivele de investiție industriale (chiar şi cele care nu sunt construite de persoane fizice).

  R = SELECT(OBIECTIV_INVESTITIE, denumire = 'industrial')

  Rezultat = OUTERJOIN(PERS_FIZICA, *R*).
- Operatorii algebrei relaționale pot fi reprezentați grafic cu ajutorul unor simboluri speciale.



Proiectarea bazelor de date

Agenda

- Justificarea proiectării bazelor de date
- Procesul de normalizare
- Forme normale
- Concluziile procesului de normalizare
- Best practice
- Exerciţii

De ce proiectarea bazelor de date?

- Prin proiectarea bazei de date se subînţelege definirea unei scheme logice care să elimine posibilitatea de apariţie a unor anomalii în procesul de exploatare
- Anomaliile care apar în lucrul cu baza de date se produc datorită dependenţelor care există între datele din cadrul relaţiilor bazei de date
- De obicei dependenţele sunt plasate greşit în tabele.
- Exemplu: "*toate avioanele cu acelaşi nume au aceeaşi capacitate*"

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

De ce proiectarea bazelor de date?

- Din cauza dependenţei introduse pot exista:
  - anomalii la inserare
  - modificare sau ştergere
  - redundanţă în date
  - probleme de reconexiune

- **Redundanţă logică**. Perechea (AIRBUS, 250) apare de trei ori.
- **Anomalie la insert**. S-a cumpărat un B727 cu 150 locuri, poate fi inserat doar dacă se defineşte o nouă valoare pentru cheia primară.
- **Anomalie la ştergere**. Dacă este ştearsă linia cu id-ul 4, atunci se pierde informaţia că un avion CAR are capacitatea 100.
- **Anomalie la modificare**. Dacă se modifică capacitatea lui B707 de la 150 la 170, atunci costul modificării este mare pentru a modifica toate înregistrările.

De ce proiectarea bazelor de date?

- **Problema reconexiunii**. Considerăm schemele relaţionale:

  AVION1 = PROJECT(AVION, A#, nume)
  AVION2 = 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

Procesul de normalizare

- Normalizarea este **procesul reversibil** de transformare a unei relaţii, în relaţii de structură mai simplă. Procesul este reversibil în sensul că **nicio informaţie nu este pierdută în timpul transformării**.

- O relaţie este într-o formă normală particulară dacă ea satisface o mulţime specificată de constrângeri

- Normalizarea are drept scop:
  - suprimarea redundanţei logice
  - evitarea anomaliilor la reactualizare
  - rezolvarea problemei reconexiunii

Procesul de normalizare

- 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 R1, R2, ..., Rk, astfel încât dependenţele lui R sunt echivalente cu reuniunea dependenţelor lui R1, R2, ..., Rk.
  - **Descompuneri fără pierderi de informaţie (L-join)**. Această descompunere presupune desfacerea relaţiei R într-o mulţime de proiecţii R1, R2, ..., Rj, astfel încât pentru orice realizare a lui R este adevărată relaţia:

$$R = JOIN(\Pi B1 (R), \Pi B2 (R), ...,\Pi Bj (R))$$

Forme normale

- 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:

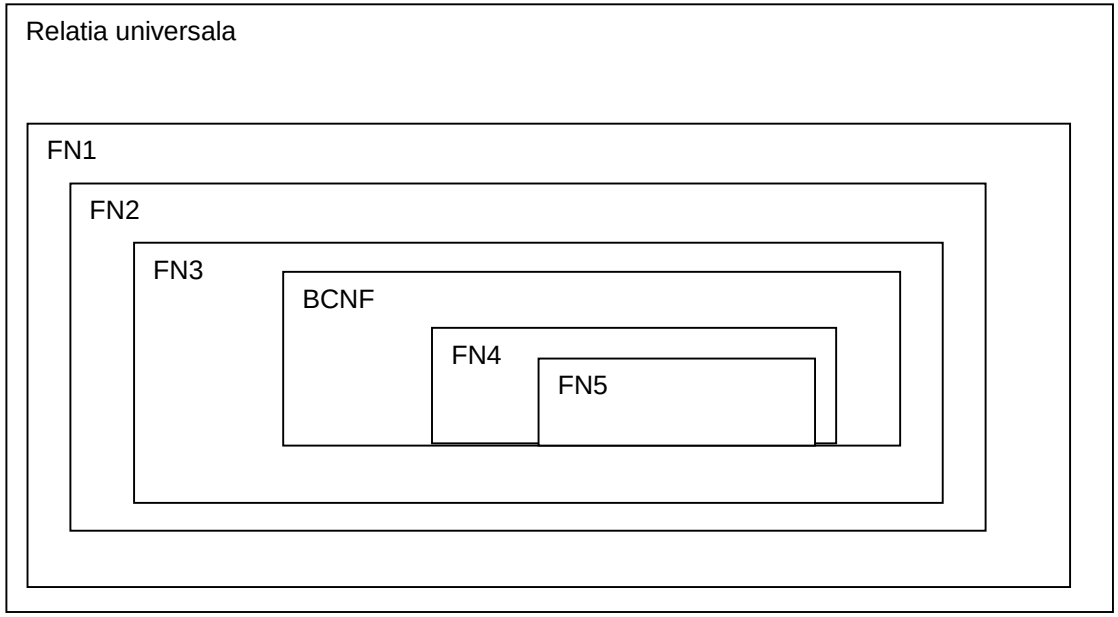
$R(A) = \text{JOIN}(\Pi_{\alpha\cup\beta}(R), \Pi_{\alpha\cup\gamma}(R))$.

$\alpha\cup\beta$ → *mulţimea atributelor care intervin în dependenţele funcţionale*;

$\alpha\cup\gamma$ → *reprezintă reuniunea determinantului cu restul atributelor lui A*.

Forme normale

- Procesul prin care o relaţie universală este trecută prin toate formele normale poate fi reprezentat astfel:



Forme normale (exemplu)

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 1 (FN1)

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

Persoana	Laptop
Mihai	R25 - W14 - R21
Tudor	205
Elena	R5 - 305
Ninel	BX - 305 - R12 - R25

Persoana	Laptop
Mihai	R25
Mihai	W14
Mihai	R21
Tudor	205
Elena	R5
Elena	305
Ninel	BX
Ninel	305
Ninel	R12
Ninel	R25

Forma normală 2 (FN2)

- O relaţie este în a doua formă normală dacă şi numai dacă:
  - relaţia R este în FN1;
  - fiecare atribut care nu este cheie este dependent de cheie primară.

Cod_salariat#	Job_cod	Nr_proiect#	Functia	Sal
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

Cod_salariat#	Job_cod
S1	Programator
S3	Vanzator
S5	Inginer

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

Forma normală 3 (FN3)

- O relaţie este în a treia formă normală dacă şi numai dacă:
  - relaţia R este în FN2;
  - fiecare atribut care nu este cheie este dependent de cheie primară, **depinde de cheie, de întreaga cheie şi numai de cheie** (elimină dependenţele tranzitive);
  - Pentru a obţine o relaţie FN3 se aplica regula Casey-Delobel pentru fiecare dependenţă tranzitivă

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

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

Functia	Sal
Supervizor	60
Cercetator	25
Auxiliar	10

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.

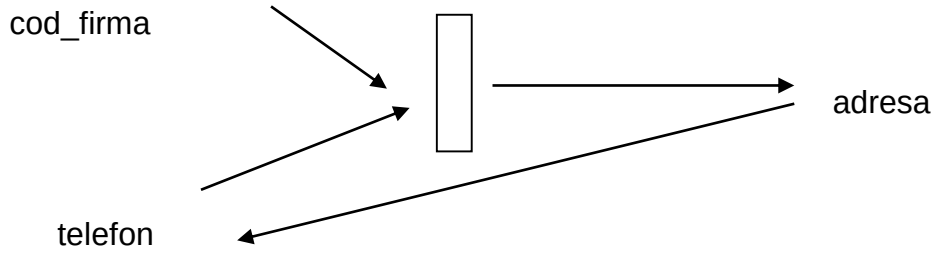
- **Intuitiv**, o relaţie R este în forma normală Boyce-Codd dacă şi numai dacă fiecare determinant este o cheie candidat.

- **Formal**, o relaţie R este în forma normală Boyce-Codd dacă şi numai dacă pentru orice dependenţă funcţională totală X → 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)

Forma normală Boyce-Codd (BCNF)

- Exemplu:
- ADRESA(cod_firma#, telefon#, adresa)



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

ADRESA_1(cod_firma#, adresa);

ADRESA_2(adresa#, telefon).

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

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

- 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)
- **Formal**, relaţia R este în a patra formă normală dacă şi numai dacă:
  - R este în BCNF;
  - orice dependenţă multivaloare este o dependenţă funcţională

- Aducerea relaţiilor în FN4 presupune eliminarea dependenţelor multivaloare atunci când sunt mai mult de una în cadrul unei relaţii

Forma normală 4 (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 relației există multidependenţele:

- cod_contractant# →→ denumire;
- cod_contractant# →→ telefon.

Relaţia INVESTITIE este în BCNF. Pentru a aduce relația 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
- **Intuitiv**, o relaţie R este în forma normală 5 dacă şi numai dacă:
  - relaţia este în FN4;
  - nu conţine dependenţe ciclice.

- Există relaţii care nu pot fi descompuse în două relaţii dar care pot fi descompuse în trei, patru sau mai multe relaţii fără a pierde informaţii. Pentru a obţine aceste descompuneri se introduce conceptul de *join*-dependenţă sau dependenţă la compunere (JD).

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

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

Forma normală 5 (FN5)

- Exemplu:

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

- relaţia este în FN4
- există redundanţă
- relaţia poate fi descompusă

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

R1(furnizor#, cod_consumabil#)

R2(furnizor#, cantitate, pret)

R3(cod_consumabil#, cantitate, pret).

- 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

Concluzii normalizare

- FN1 → FN2 elimină redundanţele datorate dependenţei netotale a atributelor care nu participă la o cheie, faţă de cheile lui *R*
- FN2 → FN3 elimină redundanţele datorate dependenţei tranzitive
- FN3 → BCNF elimină redundanţele datorate dependenţei funcţionale
- BCNF → FN4 elimină redundanţele datorate multidependenţei
- FN4 → FN5 elimină redundanţele datorate dependentei ciclice
- 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
- *În practică de cele mai multe ori se consideră suficient ca o bază de date să fie adusă în FN3*

Best practice

Proiectarea conceptuală a bazei de date:

a) identificarea tipurilor de entităţi (E);

b) identificarea tipurilor de relaţii (R);

c) identificarea şi asocierea atributelor (A) cu tipurile de E sau R;

d) determinarea domeniilor atributelor;

e) determinarea atributelor chei candidat şi chei primare;

f) specializarea şi generalizarea tipurilor de entităţi;

g) desenarea diagramei E/R;

h) revizuirea modelului de date conceptual local.

Best practice

- Proiectarea logică a bazei de date - transpunerea reprezentării conceptuale în structura logică a BD:

1. transpunerea modelului conceptual local în modelul de date logic local:
    - eliminarea relaţiilor M:N (înlocuirea prin relaţii de tip 1:M sau prin tabele asociative);
    - eliminarea relaţiilor complexe (înlocuirea prin relaţii de tip 1:M);
    - eliminarea relaţiilor recursive (înlocuirea prin relaţii dependente);
    - eliminarea atributelor multiple (înlocuirea prin relaţii dependente);
    - reexaminarea relaţiilor 1:1 (sunt într-adevăr 2 relaţii distincte?);
    - eliminarea relaţiilor redundante.

Best practice

2) extragerea relaţiilor din modelul de date logic local pentru a reprezenta entităţile şi relaţiile (legăturile);
3) normalizarea relaţiilor;
4) validarea modelului conform tranzacţiilor utilizatorului pentru a garanta că acesta acceptă şi rezolvă operaţiile cerute de către model;
5) desenarea diagramei conceptuale;
6) definirea constrângerilor de integritate;
7) revizuirea modelului de date conceptual local, împreună cu utilizatorii;
8) construirea şi validarea modelului de date logic global prin combinarea modelelor locale:
    - normalizarea modelului;
    - validarea modelului conform tranzacţiilor utilizatorului;
    - verificarea în vederea dezvoltării ulterioare;
    - desenarea diagramei conceptuale finale;
    - revizuirea modelului de date conceptual global.

Best practice

- Proiectarea reprezentării fizice

1. Pentru măsurarea eficienţei poate fi analizată:
    a) capacitatea de stocare pe disc;
    b) timpul de răspuns;
    c) transferul de tranzacţii (numărul de tranzacţii care pot fi efectuate într-un anumit interval de timp)

2. Pentru fiecare tranzacţie trebuie determinat:
    a) frecvenţa estimată cu care va fi rulată;
    b) relaţiile şi atributele accesate;
    c) tipul de acces (inserare, interogare, ştergere sau rectualizare);
    d) atributele care apar în predicate (condiţii);
    e) atributele care apar în *join*-uri;
    f) constrângerile de timp impuse tranzacţiilor.

Exerciţii

1. Sa se furnizeze un exemplu de schemă (alta decât cele descrise în curs), în care se manifesta anomalii de inserare, ștergere si modificare a datelor.

2. Să se prezinte un exemplu de relație ce se descompune fără pierderi de informații în trei relații, dar care nu se poate descompune în două (schemele sunt considerate diferite).

3. Dați un exemplu de schema în forma normala Boyce-Codd, dar care nu se găsește în forma normala patru.

Proiectarea bazelor de date

Introducere

Plan

- Introducere
- Gestiunea bazelor de date
- Arhitectura sistemelor de gestiune a bazelor de date
- Evoluția bazelor de date
- Arhitecturi multi-user pentru sisteme de gestiune a bazelor de date

Introducere (1)

- Informația are valoare doar dacă ea influențează procesul de luare a deciziilor şi determină alegerea unor decizii mai bune decât cele care ar fi luate în lipsa informației.
- Informația trebuie să fie disponibilă în timp util, să fie corectă, necontradictorie, neredondantă şi să aibă o formă adecvată necesităților factorului de decizie.
- volum imens de date care trebuie culese, memorate, organizate, regăsite şi prelucrate în mod corespunzător pentru utilizarea lor ca informație.
- O astfel de activitate este legată, în informatică, de noțiunea de bază de date.

Introducere (2)

- **Baza de date** este un ansamblu structurat de date coerente, fără redundanță inutilă, astfel încât acestea pot fi prelucrate eficient de mai mulți utilizatori într-un mod concurent.
- Baza de date este o colecție de date persistente, care sunt folosite de către sistemele de aplicații ale unei anumite „întreprinderi“.
  - Datele persistă deoarece, după ce au fost acceptate de către sistemul de gestiune pentru introducerea în baza de date, ele pot fi şterse din bază numai printr-o cerere explicită adresată sistemului de gestiune.
- Termenul de „întreprindere“ este un cuvânt generic, utilizat pentru a desemna orice organizație independentă, de natură comercială, tehnică, ştiințifică sau de alt tip.
  - Întreprinderea poate fi, de exemplu, un spital, o bancă, o facultate, o fabrică, un aeroport etc. Fiecare întreprindere are regulile proprii de funcționare şi conține o mulțime de date referitoare la modul său de operare.

Introducere (3)

- În general, întreprinderile întreţin două clase mari de date.
  - date operaţionale - în acest caz accentul este pus pe aplicaţii operaţionale (de producţie), adică aplicaţii de rutină, care reflectă funcţionalitatea zilnică a întreprinderii (de exemplu, depunerea sau extragerea banilor într-un, respectiv dintr-un sistem bancar)
  - date pentru susţinerea procesului decizional (depozit de date) -informaţii de sinteză (medii, dispersii, totaluri), care sunt extrase la anumite intervale de timp din baza de date operaţionale.
- Datele din baza de date pot fi atît integrate, cât şi partajate.
  - Noţiunea de integrat se referă la faptul că baza de date poate fi considerată ca o unificare a mai multor fişiere
  - Prin partajare se înţelege că baza de date poate fi partajată concurent între diferiţi utilizatori.

Introducere (4)

- Caracteristica principală a aplicațiilor de baze de date: accentul este pus pe operațiile de memorare şi regăsire efectuate asupra unor volume mari de date, şi mai puțin asupra operațiilor de prelucrare a acestora.
  - Principala operație care apare în orice aplicație de baze de date este aceea de regăsire a datelor, în scopul obținerii de informații din baza de date.
  - Alături de operațiile de regăsire apar, mai mult sau mai puțin frecvent, operații de memorare pentru introducerea de noi date în baza de date, operații de ştergere a datelor devenite inutile, operații de actualizare a unor date existente deja în baza de date.
- Un **sistem de gestiune a bazelor de date** (SGBD – *Data Base Management System*) este un produs *software* care asigură interacțiunea cu o bază de date, permițând definirea, consultarea şi actualizarea datelor din baza de date. Toate cererile de acces la baza de date sunt tratate şi controlate de către SGBD.

Introducere (5)

- Organizarea datelor în baze de date constituie o formă de centralizare a acestora.
- Aceasta implică existența unui **administrator al bazei de date** (DBA – *Data Base Administrator*) care este o persoană sau un grup de persoane ce răspund de ansamblul activităților (analiză, proiectare, implementare, exploatare, întreținere etc.) legate de baza de date.
  - Atribuțiile unui administrator pot fi grupate în patru mari categorii: atribuții de proiectare, atribuții administrative, atribuții operative şi atribuții de coordonare.
- **Dicționarul datelor** (catalog de sistem), structurat şi administrat ca o bază de date (metabază de date), conține „date despre date", furnizează descrierea tuturor obiectelor unei baze de date, starea acestor obiecte, diversele constrângeri de securitate şi de integritate, informații despre utilizatori etc.

Gestiunea bazelor de date (1)

- Un sistem de baze de date presupune următoarele componente principale, care definesc **arhitectura** acestuia:
  - baza de date propriu-zisă în care se memorează datele;
  - sistemul de gestiune a bazei de date, care realizează gestionarea şi prelucrarea complexă a datelor;
  - un dicționar al bazei de date (metabaza de date), ce conține informații despre date, structura acestora, statistici, documentație;
  - mijloace *hardware* (comune sau specializate);
  - reglementări administrative destinate bunei funcționări a sistemului;
  - personalul implicat (utilizatori finali, administratorul datelor, administratorul bazei de date, proiectanți, programatori de aplicații).
- Patru categorii de persoane implicate în mediul bazelor de date:
  - administratori de date şi baze de date,
  - proiectanți (designeri) de baze de date,
  - programatori de aplicații,
  - utilizatori finali.

Gestiunea bazelor de date (2)

- **Administratorul de date** (DA – *Data Administrator*) este un manager, nu un tehnician, care:
  - decide care date trebuie stocate în baza de date;
  - stabileşte regulile de întreţinere şi de tratare a acestor date după ce sunt stocate. De exemplu, o regulă ar putea fi aceea prin care se stabilesc pentru utilizatori privilegii asupra informaţiilor din baza de date, cu alte cuvinte o anumită politică de securitate a datelor.
- **Administratorul bazei de date** (DBA – *Database Administrator*) este responsabil cu implementarea deciziilor administratorului de date şi cu controlul general al sistemului, la nivel tehnic. El este un profesionist în domeniul IT, care:
  - creează baza de date reală;
  - implementează elementele tehnice de control;
  - este responsabil cu asigurarea funcţionării sistemului la performanţe adecvate, cu monitorizarea performanţelor;
  - furnizează diverse servicii tehnice etc.

Gestiunea bazelor de date (3)

- **Proiectanții de baze de date** pot acoperi atât aspectul fizic, cât şi cel logic al concepției.
- Proiectantul de baze de date care abordează direcția logică trebuie să posede o cunoaştere completă şi amănunțită a modelului real de proiectat şi a regulilor de funcționare a acestuia.
  - proiectează conceptual baza de date, iar modelul creat este independent de programele de aplicații, de limbajele de programare.
  - de asemenea, va proiecta logic baza de date, proiectare care este îndreptată spre un anumit model de date (relațional, orientat obiect, ierarhic etc.).
- Proiectantul de baze de date fizice preia modelul logic de date şi stabileşte cum va fi realizat fizic. Acesta trebuie să cunoască funcționalitățile SGBD-ului, avantajele şi dezavantajele fiecărei alternative corespunzătoare unei implementări.
  - se face transpunerea modelului logic într-un set de tabele supuse unor constrângeri, se selectează structuri de stocare şi metode de acces specifice, astfel încât să se asigure performanțe, se iau măsuri privind securitatea datelor.

Gestiunea bazelor de date (4)

- **Utilizatorii finali** sunt cei care accesează interactiv baza de date. Aceasta a fost proiectată, implementată, întreținută pentru a satisface necesitățile informaționale ale clienților.
  - Utilizatorii finali pot fi utilizatori simpli, care nu cunosc nimic despre baza de date, despre SGBD, dar accesează baza prin intermediul unor programe de aplicație. În general, această clasă de utilizatori alege anumite opțiuni din meniul aplicației.
  - Există utilizatori finali sofisticați, care sunt familiarizați cu structura bazei de date. Ei pot utiliza limbaje speciale pentru a exploata posibilitățile oferite de baza de date.
- **Programatorii de aplicații** sunt responsabili de scrierea programelor aplicație ce conferă funcționalitatea cerută de utilizatorii finali. Programele pot fi scrise în diferite limbaje de programare (C++, *PL/SQL*, *Java* etc.).

Gestiunea bazelor de date (5)

- **Cerințele minimale** care se impun unei baze de date sunt:
  - asigurarea unei redundanțe minime în date;
  - furnizarea în timp util a informațiilor solicitate (optimizarea timpului de răspuns la o interogare);
  - asigurarea unor costuri minime în prelucrarea şi întreținerea informației;
  - capacitatea de a satisface, cu aceleaşi date, necesități informaționale ale unui număr mare de utilizatori,
  - posibilitatea de adaptare la cerințe noi, răspunsuri la interogări neprevăzute inițial (flexibilitate);
  - exploatarea simultană a datelor de către mai mulți utilizatori (sincronizare);
  - asigurarea securității datelor prin mecanisme de protecție împotriva accesului neautorizat (confidențialitate);
  - înglobarea unor facilități destinate validării datelor şi recuperării lor în cazul unor deteriorări accidentale, garantarea (atât cât este posibil) că datele din baza de date sunt corecte (integritate);
  - posibilitatea de valorificare a eforturilor anterioare şi anticiparea nevoilor viitoare (compatibilitate şi expandabilitate);
  - permisivitatea, prin ierarhizarea datelor după criteriul frecvenței acceselor, a unor reorganizări (eventual dinamice) care sporesc performanțele bazei.

Gestiunea bazelor de date (6)

- În cadrul unei baze de date putem vorbi de patru **niveluri de abstractizare şi de percepție** a datelor: intern, conceptual, logic şi extern. Datele există doar la nivel fizic, iar celelalte trei niveluri reprezintă virtualizări ale acestora.
  - Nivelul fizic (intern) este descris de schema fizică a datelor (bit, octet, adresă);
  - Nivelul conceptual este descris de schema conceptuală a datelor (articol, înregistrare, zonă) şi reprezintă viziunea programatorilor de sistem asupra datelor;
  - Nivelul logic este descris de una din schemele logice posibile ale datelor şi reprezintă viziunea programatorului de aplicație asupra datelor;
  - Nivelul virtual (extern) reprezintă viziunea utilizatorului final asupra datelor.

Gestiunea bazelor de date (7)

- Unul dintre avantajele incontestabile ale sistemelor de baze de date este **independența datelor** care cuprinde două aspecte fundamentale: o modificare a structurii fizice nu va afecta aplicatia şi reciproc, modificări ale aplicației vor lăsa nealterată structura fizică de date.
  - **Independența fizică**: posibilitatea modificării schemei fizice a datelor fără ca aceasta să implice modificarea schemei conceptuale, a schemei logice şi a programelor de aplicație. Prin urmare, este vorba despre imunitatea programelor de aplicație față de modificările modului în care datele sunt stocate fizic şi accesate.
  - **Independența logică**: posibilitatea modificării schemei conceptuale a datelor fără ca aceasta să implice modificarea schemei logice şi a programelor de aplicație. Prin independența logică a datelor se urmărește a se crea fiecărui utilizator iluzia că este singurul beneficiar al unor date pe care, în realitate, le foloseşte în comun cu alți utilizatori.

Gestiunea bazelor de date (8)

- Independența față de strategiile de acces permite programului să precizeze data pe care dorește să o acceseze, dar nu modul cum accesează această dată. SGBD-ul va stabili drumul optim de acces la date.
- În limbajele de programare uzuale, declarațiile și instrucțiunile executabile aparțin aceluiași limbaj. În lumea bazelor de date, funcțiile de declarare și de prelucrare a datelor sunt realizate cu ajutorul unor limbaje diferite, numite **limbaje pentru baze de date.**

Gestiunea bazelor de date (9)

- **Limbaje pentru definirea datelor** (LDD – *Data Definition Language*). Descrierea concretă a unui LDD este specifică fiecărui sistem de gestiune, dar funcțiile principale sunt aceleaşi.
  - La nivel conceptual, LDD realizează definirea entităților şi a atributelor acestora prin: nume, formă de memorare, lungime.
  - Sunt precizate relațiile dintre date şi strategiile de acces la ele, sunt stabilite criterii diferențiate de confidențialitate, sunt definite criterii de validare automată a datelor utilizate.

Gestiunea bazelor de date (10)

- **Limbaje pentru prelucrarea datelor** (LMD – *Data Manipulation Language*).
  - Operațiile executate în cadrul unei baze de date presupun existența unui limbaj specializat, în care comenzile se exprimă prin fraze ce descriu acțiuni asupra bazei.
  - În general, o comandă are următoarea structură: operația (calcul aritmetic sau logic, editare, extragere, deschidere-închidere, adăugare, ştergere, căutare, reactualizare etc.), criterii de selecție, mod de acces (secvențial, indexat etc.), format de editare.
  - Există limbaje LMD procedurale, care specifică modul în care se obține rezultatul unei comenzi LMD şi limbaje neprocedurale, care descriu doar datele ce vor fi obținute şi nu modalitatea de obținere a acestora.

Gestiunea bazelor de date (11)

- **Limbaje pentru controlul datelor** (LCD – *Data Control Language*).
  - Controlul unei baze de date se referă la asigurarea confidențialității şi integrității datelor, la salvarea informației în cazul unor defecțiuni, la obținerea unor performanțe, la rezolvarea unor probleme de concurență.

Gestiunea bazelor de date (12)

- Limbajele universale nu se utilizează frecvent pentru gestionarea unei baze de date, dar există această posibilitate.
  - De exemplu, sistemul *Oracle* este dotat cu precompilatoare (*C/C++*, *Pascal*, *ADA*, *Cobol*, *PL/1*, *Fortran*) care ajută la incorporarea de instrucțiuni *SQL* sau blocuri *PL/SQL* în programe scrise în alte limbaje, de nivel înalt, numite limbaje gazdă.
- Sistemul de gestiune a bazelor de date interacționează cu programele de aplicație ale utilizatorului şi cu baza de date, oferind o mulțime de facilități.
- Realizarea optimă a acestor facilități este asigurată de **obiectivele** fundamentale ale unui sistem de gestiune. Câteva dintre aceste obiective vor fi enumerate în continuare.

Gestiunea bazelor de date (13)

- **Independența fizică**. Obiectivul esențial este acela de a permite realizarea independenței structurilor de stocare în raport cu structurile de date din lumea reală.
  - Se defineşte mulțimea de date indiferent de forma acesteia din lumea reală, ținând seama doar de realizarea unui un acces simplu la date şi de obținerea anumitor performanțe.
- **Independența logică**. Grupul de lucru care exploatează baza de date poate să utilizeze diferite informații de bază (nu aceleaşi) pentru a-şi construi entități şi relații.
  - Fiecare grup de lucru poate să cunoască doar o parte a semanticii datelor, să vadă doar o submulțime a datelor şi numai sub forma în care le doreşte.
  - Această independență asigură imunitatea schemelor externe față de modificările făcute în schema conceptuală.

Gestiunea bazelor de date (14)

- **Prelucrarea datelor de către neinformaticieni**. Neinformaticienii văd datele independent de implementarea lor şi pot exploata aceste date prin intermediul unor acțiuni oferite de aplicația pe care o exploatează.
- **Administrarea centralizată a datelor**. Administrarea datelor presupune definirea structurii datelor şi a modului de stocare a acestora. Administrarea este în general centralizată şi permite o organizare coerentă şi eficace a informației.
- **Neredundanța datelor**. Fiecare aplicație posedă datele sale proprii şi aceasta conduce la numeroase dubluri. De asemenea, organizarea nejudicioasă a relațiilor poate să genereze redundanță în date.
  - Administrarea coerentă a datelor trebuie să asigure neduplicarea fizică a datelor. Totuşi, nu sunt excluse nici cazurile în care, pentru a realiza performanțe referitoare la timpul de acces la date şi răspuns la solicitările utilizatorilor, se acceptă o anumită redundanță a datelor.

Gestiunea bazelor de date (15)

- **Coerența datelor**. Informația trebuie să satisfacă anumite constrângeri statice sau dinamice, locale sau generale.
  - De exemplu, pot fi considerate drept constrângeri: apartenența la un anumit domeniu, un anumit tip de date, de o anumită lungime, o anumită cardinalitate a relațiilor, constrângeri referențiale.
- **Partajabilitatea datelor.** Aceasta permite ca aplicațiile să partajeze datele din baza de date în timp şi simultan.
  - O aplicație poate folosi date ca şi cum ar fi singura care le utilizează, fără a şti că altă aplicație, în mod concurent, le poate modifica.

Gestiunea bazelor de date (16)

- **Securitatea şi confidenţialitatea datelor**. Datele trebuie protejate de un acces neautorizat sau rău intenţionat.
  - Există mecanisme care permit identificarea şi autentificarea utilizatorilor şi există proceduri de acces autorizat care depind de date şi de utilizator.
  - Sistemul de gestiune trebuie să asigure securitatea fizică şi logică a informaţiei şi să garanteze că numai utilizatorii autorizaţi pot efectua operaţii asupra bazei de date.

Gestiunea bazelor de date (17)

- Sistemele de gestiune a bazelor de date au, din nefericire, şi **dezavantaje** dintre care se remarcă:
  - complexitatea şi dimensiunea sistemelor pot să crească considerabil, datorită necesității extinderii funcționalităților sistemului;
  - costul, care variază în funcție de mediu şi funcționalitatea oferită, la care se adugă cheltuieli periodice de întreținere;
  - costuri adiționale pentru elemente de *hardware*;
  - costul conversiei aplicațiilor existente, necesară pentru ca acestea să poată funcționa în noua configurație *hardware* şi *software*;
  - impactul unei defecțiuni asupra aplicațiilor, bazei de date sau sistemului de gestiune.

Gestiunea bazelor de date (18)

- **Structura** unui sistem de gestiune a bazelor de date este de complexitate variabilă, iar nivelul real de funcționalitate diferă de la un produs la altul. În orice moment apar noi necesități, care cer o nouă funcționalitate, astfel încât structura nu va putea deveni niciodată statică.
- În general, un SGBD trebuie să includă cel puțin cinci clase de module:
  - programe de gestiune a bazei de date (PGBD), care realizează accesul fizic la date ca urmare a unei comenzi;
  - module pentru tratarea limbajului de definire a datelor, ce permit traducerea unor informații (care realizează descrierea datelor, a legăturilor logice dintre acestea şi a constrângerilor la care sunt supuse) în obiecte ce pot fi apoi exploatate în manieră procedurală sau neprocedurală;
  - module pentru tratarea limbajului de prelucrare a datelor (interpretativ, compilativ, generare de programe), care permit utilizatorilor inserarea, ştergerea, reactualizarea sau consultarea informației dintr-o bază de date;
  - module utilitare, care asigură întreținerea, prelucrarea, exploatarea corectă şi uşoară a bazei de date;
  - module de control, care permit controlul programelor de aplicație, asigurarea confidențialității şi integrității datelor, rezolvarea unor probleme de concurență, recuperarea informației în cazul unor avarii sau defecțiuni *hardware* sau *software* etc.

Gestiunea bazelor de date (19)

- SGBD poate fi considerat cea mai importantă componentă *software* din sistemul de operare general, dar nu este singura.
- Celelalte componente cuprind programele utilitare, instrumentele pentru dezvoltarea aplicațiilor, programe de asistență în *design*, editoare de rapoarte, managerul de tranzacții etc

Gestiunea bazelor de date (20)

- Modulele PGBD asigură accesul fizic la date ca urmare a unei comenzi. Cum lucrează aceste module?
  - Găsesc descrierea datelor implicate în comandă.
  - Identifică datele şi tipul acestora.
  - Identifică informații ce permit accesul la structurile fizice de stocare (fişiere, volume etc.).
  - Verifică dacă datele sunt disponibile.
  - Extrag datele, fac conversiile, plasează datele în spațiul de memorie al utilizatorului.
  - Transmit informații de control necesare execuției comenzii, în spațiul de memorie al utilizatorului.
  - Transferă controlul programului de aplicație.

Gestiunea bazelor de date (21)

- **Prin urmare, din punct de vedere conceptual:**
  - **utilizatorul lansează o cerere de acces;**
  - **SGBD-ul acceptă cererea şi o analizează;**
  - **SGBD-ul, inspectează pe rând, schema internă corespunzatoare utilizatorului, schema conceptuală, definiția structurii de stocare şi corespondențele corespunzătoare;**
  - **SGBD-ul execută operațiile necesare în baza de date stocată.**

Arhitectura sistemelor de gestiune a bazelor de date (1)

- Asigurarea independenței fizice şi logice a datelor impune adoptarea unei arhitecturi de baze de date organizată pe trei niveluri:
  - **nivelul intern** (baza de date fizică);
  - **nivelul conceptual** (modelul conceptual, schema conceptuală);
  - **nivelul extern** (modelul extern, subschema, vizualizarea).

Arhitectura SGBD (2)

- Nivelul central este nivelul conceptual.
  - Acesta corespunde structurii canonice a datelor ce caracterizează procesul de modelat, adică structura semantică a datelor fără implementarea pe calculator.
  - Schema conceptuală permite definirea tipurilor de date ce caracterizează proprietățile elementare ale entităților, definirea tipurilor de date compuse care permit regruparea atributelor pentru a descrie entitățile modelului şi legăturile între aceste entități, definirea regulilor pe care trebuie să le respecte datele etc.

Arhitectura SGBD (3)

- Nivelul intern corespunde structurii interne de stocare a datelor.
  - Schema internă permite descrierea datelor unei baze sub forma în care sunt stocate în memoria calculatorului.
  - Sunt definite fişierele care conțin aceste date, articolele din fişiere, căile de acces la aceste articole etc.

Arhitectura SGBD (4)

- La nivel conceptual sau intern, schemele descriu o bază de date.
- La nivel extern schemele descriu doar o parte din date care prezintă interes pentru un utilizator sau un grup de utilizatori.
  - Schema externă reprezintă o descriere a unei părți a bazei de date ce corespunde viziunii unui program sau unui utilizator.
  - Modelul extern folosit este dependent de limbajul utilizat pentru prelucrarea bazei de date.
  - Schema externă permite asigurarea unei securități a datelor. Un grup de lucru va accesa doar datele descrise în schema sa externă, iar restul datelor sunt protejate împotriva accesului neautorizat sau rău intenționat.
- Pentru o bază de date particulară există o singură schemă internă, o singură schemă conceptuală, dar există mai multe scheme externe.

Arhitectura SGBD (5)

- În afară de aceste trei niveluri, arhitectura presupune şi anumite corespondențe între acestea:
  - corespondența conceptual-intern defineşte relația dintre nivelul conceptual şi baza de date stocată, specificând modul în care înregistrările şi câmpurile conceptuale sunt reprezentate la nivel intern;
  - corespondența extern-conceptual defineşte relația dintre o anumită vizualizare externă şi nivelul (vizualizarea) conceptual, reprezentând cheia independenței logice relativ la date;
  - corespondența extern-extern permite definirea unor vizualizări externe în funcție de altele, fără a necesita o definiție explicită a corespondenței cu nivelul conceptual.

Arhitectura SGBD (6)

- **Arhitectura funcțională de referință** propusă de grupul de lucru ANSI/X3/SPARC este axată pe dicționarul datelor şi cuprinde două părți:
  - prima, permite descrierea datelor (compoziția dicționarului datelor);
  - a doua, permite prelucrarea datelor (interogarea şi reactualizarea bazei de date).
- În fiecare parte se regăsesc cele trei niveluri: intern, conceptual şi extern. Acestea nu sunt neapărat distincte pentru orice SGBD.

Arhitectura SGBD (7)

- G. Gardarin a propus o arhitectură funcțională, apropiată de arhitectura sistemelor de gestiune actuale, care are la bază doar două niveluri:
  - schema, care corespunde integrării schemelor interne şi conceptuale;
  - vizualizarea, care este o schemă externă.
- Sistemul de gestiune gestionează un **dicționar de date** care este alimentat prin comenzi de definire a schemei şi prin comenzi de definire a vizualizărilor.
- Aceste comenzi, precum şi cererile de prelucrare, sunt analizate şi tratate de un procesor numit **analizor**. Analizorul realizează analiza sintactică şi semantică a cererii şi o traduce în format intern. O cerere în format intern care face referință la o vizualizare este tradusă în una sau mai multe cereri care fac referință la obiecte ce există în baza de date (modificarea cererilor).

Arhitectura SGBD (8)

- În cadrul acestei arhitecturi există un procesor, numit **translator**, care realizează modificarea cererilor, asigură controlul drepturilor de acces şi controlul integrității în cazul reactualizărilor.
- Componenta cheie a sistemului de gestiune este procesorul **optimizor** care elaborează un plan de acces optim pentru a trata cererea. Acest procesor descompune cererea în operații de acces elementare şi alege o ordine de execuție optimală. De asemenea, evaluează costul planului de acces înaintea execuției sale.
- Planul de acces ales şi elaborat de optimizor este executat de un procesor numit **executor**. La acest nivel este gestionat controlul concurenței.

Evoluția bazelor de date (1)

- Istoria bazelor de date şi a sistemelor de gestiune a bazelor de date poate fi rezumată în trei generaţii: sisteme ierarhice şi reţea, sisteme relaţionale şi sisteme avansate (orientate obiect, relaţionale orientate obiect, deductive, distribuite, multimedia, multibaze, active, temporale, decizionale, magazii de date etc.).

Baze de date ierarhice şi reţea

- Pentru modelele ierarhice şi reţea, datele sunt reprezentate la nivel de articol prin legături ierarhice (arbore) sau de tip graf. Slaba independenţă fizică a datelor complică administrarea, reactualizarea şi prelucrarea acestora. Limbajul de prelucrare a datelor impune programatorului să specifice căile de acces la date.
- Structurile de date corespunzătoare acestor modele pot fi descrise la nivel logic în termenii unui instrument de abstractizare, numit diagrama structurii de date. Diagrama este un graf orientat reprezentând tipuri de entităţi şi legături funcţionale între acestea.

Evoluția bazelor de date (2)

Baze de date relaționale

- A doua generație de SGBD-uri este legată de apariția modelelor relaționale (1970), care tratează entitățile ca niște relații. Piața actuală de baze de date este acoperită în majoritate de sisteme **relaționale**. Acestea, ca și modelele din prima generație, au fost concepute pentru aplicații clasice: contabilitate, gestiunea stocurilor etc. Bazele de date relaționale sunt caracterizate de:
- structuri de date simple, intuitive;
- inexistența pointerilor vizibili pentru utilizator;
- constrângeri de integritate;
- operatori aplicați relațiilor care permit definirea, căutarea şi reactualizarea datelor.

Evoluția bazelor de date (3)

- Bazele de date relaționale oferă avantaje precum:
  - independența completă în descrierea logică a datelor (în termen de relații) şi în descrierea fizică a datelor (în termen de fişiere)
  - un ansamblu integrat de utilitare bazat pe un limbaj de generația a 4-a (generatoare de meniuri, generatoare de aplicații, generatoare de forme, generatoare de etichete)
  - existența unor limbaje speciale de definire şi prelucrare a datelor.

Evoluția bazelor de date (4)

- Dezvoltarea unei aplicații riguroase, utilizând baze de date relaționale, necesită cunoaşterea a trei niveluri de instrumente, eterogene din punct de vedere conceptual:
  - nivelul instrumentelor grafice (interfața);
  - nivelul aplicației, cu limbajele sale de dezvoltare;
  - nivelul SGBD, cu standardul *SQL* (*Structured Query Language*) ce permite definirea, prelucrarea şi controlul bazei de date.

Evoluția bazelor de date (5)

Baze de date orientate obiect

- Bazele de date relaționale nu folosesc însă obiecte complexe şi dinamice, nu realizează gestiunea datelor distribuite şi nici gestiunea cunoştințelor.
- A treia generație de SGBD-uri, sistemele avansate, încearcă să depăşească aceste limite ale sistemului relațional.
- Suportul obiectelor complexe şi dinamice şi prelucrarea acestora sunt dificile pentru sistemele relaționale, deoarece tipul datelor este limitat la câteva domenii alfanumerice, iar structura datelor este simplă.
- Sistemele relaționale nu modelează obiecte complexe ca grafuri, liste etc. Un obiect complex poate să fie descompus în relații, dar apar dificultăți atât la descompunerea, cât şi la refacerea acestuia prin compunere. De asemenea, limbajele modelului relațional permit prelucrarea cu dificultate a obiectelor complexe.

Evoluția bazelor de date (6)

- O aplicație are un aspect static (reprezentat prin date) şi unul dinamic (reprezentat prin tratamentul acestor date).
- Obiectele prelucrate de sistemele relaționale sunt în general statice, iar comportamentul lor (dinamica lor) este dat separat prin programele de aplicație care le exploatează.
- Un sistem relațional nu suportă obiecte dinamice care incorporează atât partea de date (informații) efective, cât şi o parte relativă la tratarea acestora.
- În programarea orientată obiect, efortul constă în definirea obiectelor. Obiectele de acelaşi tip formează o clasă care este generalizarea noțiunii de tip de date definit de utilizator. Clasa include, pe lângă date, şi metodele de acces la ele. Datele sunt vizibile doar metodelor asociate clasei respective. Aceasta defineşte principiul încapsulării datelor. Prin funcții numite constructori şi destructori, programatorul deține controlul asupra operațiilor necesare la crearea, respectiv dispariția unui anumit obiect. Un alt concept fundamental este cel al derivării, adică poate fi definită o clasă care să moştenească proprietățile (care constau din date şi funcții) uneia sau mai multor clase „părinte“.

Evoluția bazelor de date (7)

- Îmbinarea tehnicii limbajelor orientate obiect cu cea a bazelor de date a condus la realizarea bazelor de date orientate obiect. Acestea permit organizarea coerentă a obiectelor partajate între utilizatori concurenți. Sistemele de gestiune de baze de date orientate obiect (SGBDOO) prezintă o serie de avantaje:
- realizează o modelare superioară a informației,
- furnizează posibilități superioare de deducție (ierarhie de clase, moştenire),
- permit luarea în considerare a aspectelor dinamice şi integrarea descrierii structurale şi comportamentale,
- asigură îmbunătățirea interfeței cu utilizatorul.

Evoluția bazelor de date (8)

- Cu toate avantajele incontestabile oferite de SGBDOO-uri, impunerea lor pe piața bazelor de date nu a fost uşoară. Câteva motivații ale acestei situații:
- absența unei fundamentări teoretice face imposibilă definirea unui SGBDOO de referință;
- gestiunea obiectelor complexe este mai dificilă decât accesul la relații prin cereri *SQL*;
- utilizatorii au investit sume uriaşe în sistemele relaționale şi nu le pot abandona cu uşurință. Trecerea la noua tehnologie orientată obiect implică investiții mari şi nu păstrează aproape nimic din vechile soluții.

Evoluția bazelor de date (9)

Baze de date relaționale orientate obiect

- Simplitatea modelului relațional, combinată cu puterea tehnologiei orientate obiect a generat un domeniu nou şi promițător în lumea bazelor de date, şi anume bazele de date relaționale orientate obiect.
- Construcția unui sistem de gestiune de baze de date relaționale orientate obiect (SGBDROO) trebuie să pornească de la sisteme existente. Aceasta se poate realiza în două moduri: dezvoltând un sistem relațional prin adăugarea caracteristicilor obiectuale necesare sau pornind de la un sistem orientat obiect şi adăugând caracteristicile relaționale.

Evoluția bazelor de date (10)

Baze de date deductive

- O relație este o mulțime de înregistrări ce reprezintă fapte. Cunoştinţele sunt aserțiuni generale şi abstracte asupra faptelor. Cunoştinţele permit raționamente, iar acestea au ca rezultat deducerea de noi fapte, plecând de la fapte cunoscute.
- Un SGBD relațional suportă o formă limitată de cunoştinţe, şi anume constrângerile de integritate, iar restul trebuie integrate în programele de aplicație. Aceasta generează probleme deoarece cunoştinţele trebuie codificate în programe şi apare imposibilitatea de a partaja cunoştinţe între utilizatori. Totul se complică dacă există un volum mare de fapte.

Evoluția bazelor de date (11)

- Bazele de date deductive, utilizând programarea logică, gestionează cunoştinţe relative la baze de date care, în general, sunt relaționale. Bazele de date deductive permit deducerea de noi informații, plecând de la informațiile stocate în baza de date. Un SGBD deductiv posedă:
  - un limbaj de definire a datelor, care permite definirea structurii predicatelor sub formă de relații şi constrângeri de integritate asociate;
  - un limbaj de prelucrare a datelor, care permite efectuarea reactualizărilor asupra datelor şi formularea unor cereri;
  - un limbaj de reguli de deducție, care permite ca, plecând cu predicatele definite anterior, să se specifice cum pot fi construite predicate derivate.

Evoluția bazelor de date (12)

Baze de date distribuite

- Un sistem distribuit este un ansamblu de maşini ce sunt interconectate printr-o reţea de comunicaţie şi utilizate într-un scop global.
- Administrarea şi prelucrarea datelor distribuite, situate pe diferite calculatoare şi exploatate de sisteme eterogene, reprezintă obiectivul fundamental al bazelor de date distribuite.
- Bazele de date distribuite sunt sisteme de baze de date cooperante care rezidă pe maşini diferite, în locuri diferite. Această mulţime de baze de date este exploatată de utilizator ca şi cum ar fi o singură bază de date.
- Programul de aplicaţie care exploatează o bază de date distribuite poate avea acces la date rezidente pe mai multe maşini, fără ca programatorul să cunoască localizarea datelor.

Evoluția bazelor de date (13)

- Modelul relațional a rămas instrumentul principal prin care se realizează prelucrarea datelor distribuite.
- Câteva dintre argumentele pentru a justifica această afirmație sunt:
  - bazele relaționale oferă flexibilitate de descom-punere în vederea distribuirii;
  - operatorii relaționali pot fi folosiți pentru combinații dinamice ale informațiilor descentralizate;
  - limbajele sistemelor relaționale sunt concise şi asigură o economie considerabilă a transmiterii datelor. Ele fac posibil, pentru un nod oarecare al rețelei, să analizeze intenția unei tranzacții, să o descompună rapid în componente ce pot fi realizate local şi componente ce pot fi transportate altor noduri.

Evoluția bazelor de date (14)

Baze de date cu suport decizional

- În prezent, ciclul de viață al produselor devine din ce în ce mai scurt, iar clienții așteaptă produse şi servicii noi la intervale tot mai mici.
- Pentru a rămâne competitive, afacerile trebuie să reacționeze rapid şi eficient la schimbări. În acest sens, ele trebuie să analizeze datele pe care le dețin pentru a putea prevedea tendințele pieții.
- Astăzi, sistemele informatice, în particular bazele de date, au ajuns la maturitate. Marile companii au acumulat o mare cantitate de informații din domeniul lor de activitate, pe care le păstrează în tabele istorice şi sunt nefolositoare sistemelor operaționale ale companiei, care funcționează cu date curente.
- Analizate, aceste date ar putea oferi informații despre tendințe şi evoluții care ar putea interesa compania. Pentru a putea analiza aceste mari cantități de date este nevoie de tehnologii şi instrumente speciale.

Evoluția bazelor de date (15)

- Ideea de a analiza colecții de date provenind din sistemele operaționale ale companiei sau din surse externe pentru a le folosi ca suport în procesul de decizie nu aparține ultimului deceniu, dar baze de date care să funcționeze eficient după aceste criterii au fost studiate şi implementate în ultimii ani.
- Principalul scop al acestor baze de date a fost de a întâmpina nevoile sistemelor operaționale, a căror natură este inerent tranzacțională.
- Sistemele tranzacționale sunt interesate, în primul rând, să controleze la un moment dat o singură tranzacție.
  - De exemplu, într-un sistem bancar, atunci când clientul face un depozit, sistemul operațional bancar este responsabil de a înregistra tranzacția într-un tabel al tranzacțiilor şi de a creşte nivelul curent al contului clientului, stocat în alt tabel.

Evoluția bazelor de date (16)

- Un sistem operațional tipic operează cu evenimente predefinite şi, datorită naturii lor, necesită acces rapid la date. Uzual, fiecare tranzacție operează cu cantități mici de date.
- De-a lungul timpului, nevoile sistemelor operaționale nu se schimbă mult. Aplicația care înregistrează tranzacția, ca şi cea care controlează accesul utilizatorului la informație (partea de raportare a sistemului bancar), nu se modifică prea mult.
- În acest tip de sistem, informația necesară în momentul în care un client inițiază o tranzacție trebuie sa fie actuală. Înainte ca o bancă să aprobe un împrumut este nevoie să se asigure de situația financiară stabilă a clientului în acel moment, şi nu cu un an înainte.

Evoluția bazelor de date (17)

- În ultimii ani s-au pus la punct principii şi tehnologii noi care să servească procesului de analiză şi administrare a datelor. O bază de date optimizată în acest scop defineşte o ***Data Warehouse*** (magazie de date), iar principiul pe care îl urmează este cunoscut sub numele de procesare analitică (OLAP – *On Line Analytical Processing*). Spre deosebire de aceasta, principiul pe care se bazează sistemele tranzacționale a fost numit procesare tranzacțională (OLTP – *On Line Transactional Processing*).
- Aplicațiile unei *Data Warehouse* trebuie să ofere răspunsuri unor întrebări de tipul: „Care zi din săptămână este cea mai aglomerată?“ „Ce clienți, cu care avem relații intense, nu au beneficiat de reduceri de prețuri?“. O caracteristică a bazelor de date analitice este că interogările la care acestea trebuie să răspundă sunt *ad-hoc*, nu sunt predefinite, iar baza de date trebuie optimizată astfel încât să fie capabilă să răspundă la orice fel de întrebare care poate implica mai multe tabele.

Evoluția bazelor de date (18)

- În această abordare, organele generale de decizie necesită accesul la toate datele organizației, oriunde s-ar afla acestea. Pentru o analiză corespunzătoare a organizației, afacerilor, cerințelor, tendințelor este necesară nu numai accesarea valorilor curente din baza de date, ci şi a datelor istorice. Prin urmare, pentru a facilita acest tip specific de analiză a informației a fost creată magazia de date, care conține informații extrase din diverse surse, întreținute de diverse unități operative, împreună cu istoricul şi rezumatul tranzacțiilor.
- Sursele de date pentru o magazie cuprind:
  - date operaționale, păstrate în baze de date ierarhice, de prima generație;
  - date departamentale, păstrate în sisteme de fişiere patentate;
  - date cu caracter personal, păstrate pe stații de lucru şi servere personale;
  - sisteme externe (baze de date comerciale, Internet etc.)

Evoluția bazelor de date (19)

- *Data warehouse* este o colecție de date:
  - orientate spre subiect (principalele subiecte ale modelului sunt clienții, produsele, vânzările, în loc de domeniile de activitate),
  - nevolatile (datele nu sunt reactualizate, înlocuite în timp real, ci sunt adăugate ca un supliment al bazei),
  - integrate (transpunerea datelor provenite din diverse surse de informații se face într-un format consistent),
  - variabile în timp (concentrarea depozitului de date se face asupra schimbărilor care au loc în timp).

Evoluția bazelor de date (20)

- Înmagazinarea datelor se concentrează asupra gestionării a cinci fluxuri de informații:
  - fluxul intern, care reprezintă procesele asociate extragerii şi încărcării datelor din fişierele sursă în magazia de date;
  - fluxul ascendent, care reprezintă procesele asociate adăugării de valoare datelor din magazie, cu ajutorul împachetării şi distribuirii;
  - fluxul descendent, care reprezintă procesele asociate arhivării, salvării, refacerii datelor din magazie;
  - fluxul extern, care reprezintă procesele asociate punerii la dispoziție a datelor pentru utilizatorii finali;
  - meta-fluxul, care reprezintă procesele asociate gestionării meta-datelor (date despre date).

Evoluția bazelor de date (21)

- În arhitectura depozitului de date intervin câteva componente specifice acestei structuri.
  - Administratorul pentru încărcarea datelor (componenta *front-end*) realizează toate operațiile asociate cu obținerea (extragerea) și încărcarea datelor operaționale într-un depozit de date.
  - Administratorul depozitului de date realizează toate operațiile legate de administrarea datelor din depozit. Operațiile realizate de componenta de administrare a depozitului de date includ: analiza datelor pentru a asigura consistența acestora; transformarea și mutarea datelor sursă din structurile temporare de stocare în tabelele depozitului de date; crearea de indecși și vizualizări asupra tabelelor de bază; generarea denormalizării (dacă este necesar); generarea agregărilor; crearea arhivelor şi a *backup*-urilor.
  - Administratorul cererilor (componenta *back-end*) realizează toate operațiile legate de administrarea cererilor utilizator. Această componentă este construită folosind utilitarele de acces la date disponibile utilizatorilor finali, utilitarele de monitorizare a depozitului de date, facilitățile oferite de sistemul de baze de date şi programele personalizate.

Evoluția bazelor de date (22)

- În zona ce include date agregate sunt stocate toate agregările predefinite de date, pe diferite niveluri. Scopul, menținerii acestora, este de a mări performanța cererilor care necesită agregări. Datele agregate sunt actualizate permanent, pe măsură ce sunt încărcate noi informații în depozit.
- Scopul principal al depozitelor de date este de a oferi informații necesare utilizatorilor pentru luarea deciziilor strategice de marketing. Aceşti utilizatori interacționează cu depozitul de date prin diferite utilitare de acces (utilitare pentru rapoarte şi cereri, utilitare pentru dezvoltarea aplicațiilor, utilitare pentru procesarea analitică *on-line* (OLAP), utilitare *data mining*) etc.

Evoluția bazelor de date (23)

- Instrumentele de acces pentru utilizatorii finali ai magaziilor de date sunt:
  - prelucrarea analitică *on-line*;
  - extensiile limbajului *SQL*;
  - instrumentele de extragere a datelor.
- Prelucrarea analitică *on-line* (OLAP) reprezintă sinteza, analiza şi consolidarea dinamică a unor volume mari de date multidimensionale. Serverele de baze de date OLAP utilizează structuri multidimensionale pentru stocarea datelor şi a relațiilor dintre date. Aceste structuri pot fi vizualizate prin cuburi de date şi cuburi în cadrul cuburilor etc. Fiecare latură a cubului reprezintă o dimensiune. Serverele de baze de date OLAP multidimensionale acceptă operațiile analitice uzuale: consolidarea (gruparea), parcurgerea în jos (inversul consolidării), tranşarea, tăierea. OLAP necesită o modalitate de agregare a datelor conform mai multor grupări diferite, în număr foarte mare, iar utilizatorii trebuie să le aibă în vedere pe toate.

Evoluția bazelor de date (24)

- Instrumentele OLAP presupun organizarea informației într-un model multidimensional care este susținut de o bază de date:
  - multidimensională (MOLAP), în care datele sunt stocate conceptual în celulele unui tablou multidimensional;
  - relațională (ROLAP), proiectată pentru a permite interogări multidimensionale.
- În acest context, a devenit o necesitate extinderea limbajului *SQL* prin operații puternice, necesare pentru rezolvarea noului tip de abordare. Au fost introduse noi funcții numerice (limita inferioară, limita superioară etc.), noi funcții statistice (distribuție, distribuție inversă, corelație etc.), noi operatori de agregare, extensii ale clauzei *GROUP BY* etc.

Evoluția bazelor de date (25)

- O altă problemă esențială este extragerea datelor și utilizarea acestora pentru luarea de decizii cruciale în domeniul afacerilor. Descoperirea unor noi corelații, tipare, tendințe, prin extragerea unor cantități mari de date folosind strategia inteligenței artificiale este una din modalitățile de rezolvare.
- Extragerea datelor presupune capacitatea de a construi modele mai degrabă previzibile, decât retrospective. Modelarea predictivă utilizează informații pentru a forma un model al caracteristicilor importante ale unui fenomen.

Evoluția bazelor de date (26)

- Tehnicile asociate celor patru operații fundamentale de extragere sunt:
  - modelarea predictivă (clasificarea cu ajutorul unei rețele neurale sau al unei inducții de tip arbore și previziunea valorilor, utilizând tehnici de regresie);
  - segmentarea bazei de date (comasarea demografică și comasarea neurală care se deosebesc prin metodele utilizate pentru a calcula distanța dintre înregistrări, prin intrările de date permise);
  - analiza legăturilor (descoperirea asocierilor, descoperirea tiparelor, descoperirea secvențelor de timp similare);
  - detectarea deviațiilor (statistici și vizualizări pentru identificarea împrăștierii datelor, utilizând tehnici moderne de vizualizare grafică). În această clasă pot fi considerate, de exemplu, detectarea fraudelor privind utilizarea cărților de credit, pretențiile de despăgubire ale asiguraților etc.

Evoluția bazelor de date (27)

- În concluzie, spre deosebire de un sistem OLTP, *Data Warehouse* este o bază de date a cărei structură este proiectată pentru a facilita analiza datelor.
- Un sistem de suport decizional urmăreşte, în primul rând, obținerea de informații din baza de date, în timp ce unul OLTP urmăreşte introducerea de informații în baza de date.
- Datorită acestor diferențe, structura optimă a unei *Data Warehouse* este radical diferită de cea a unui sistem OLTP.

Evoluția bazelor de date (28)

- Depozitele de date şi sistemele OLTP sunt supuse unor cerințe diferite, dintre care cele mai semnificative se referă la operații, actualizarea datelor, proiectare, operații tipice şi date istorice.
  - **Operații**. Depozitele sunt create pentru a permite interogări *ad hoc*. Ele trebuie să fie suficient de flexibile pentru a putea răspunde interogărilor spontane ale utilizatorilor. Sistemele OLTP suportă numai operații predefinite. Aplicațiile pot fi optimizate sau create special numai pentru acele operații.
  - **Actualizarea datelor**. Utilizatorii finali ai unui depozit de date nu fac în mod direct actualizări ale depozitului. In sistemele OLTP, utilizatorii realizează, de obicei, în mod individual procedurile de modificare a bazei de date. In acest fel, baza de date OLTP este întotdeauna la zi şi reflectă starea curentă a fiecărei tranzacții.

Evoluția bazelor de date (29)

- **Proiectare**. Depozitele de date folosesc, în mod uzual, scheme denormalizate, în timp ce sistemele OLTP folosesc scheme normalizate pentru a optimiza performanțele operațiilor.
- **Operații tipice**. O simplă interogare a depozitului de date poate scana mii sau chiar milioane de înregistrări (de exemplu, cererea „Care sunt vânzările totale către toți clienții din luna trecută?"), în timp ce o operație tipică OLTP accesează doar o parte mai mică din înregistrări.
- **Date istorice**. Depozitele de date stochează datele pe o perioadă lungă de timp, luni sau ani. Acest lucru oferă suport pentru analiza istorică a informației. Sistemele OLTP rețin date istorice atât timp cât este necesar pentru a îndeplini cu succes cerințele tranzacțiilor curente.

Evoluția bazelor de date (30)

- O bază de date OLAP poate fi relațională, dar datorită naturii ei orientate spre dimensiuni, au fost dezvoltate pentru gestionarea acestor baze de date construcții multidimensionale, mai potrivite pentru o raportare flexibilă. Spre deosebire de bazele de date relaționale, structura unei baze de date multidimensionale nu implică tabele, linii și coloane, ci obiecte de următoarele tipuri: variabile, dimensiuni, niveluri, ierarhii, atribute.
- *Data Warehouse*, care cuprinde de obicei informații despre o întreagă companie, poate fi subîmpărțită în baze de date departamentale, numite rafturi de date (*Data Marts*). De exemplu, poate exista un *Data Mart* al departamentului financiar, un altul al departamentului vânzări și un altul pentru departamentul marketing.

Evoluția bazelor de date (31)

- În ultimii ani, marii producători de sisteme de gestiune a bazelor de date relaționale, precum *Oracle*, au introdus în produsele lor construcții care să faciliteze accesul la datele din sistemele fundamentale pentru luarea de decizii.
- Astfel, noile versiuni de SGBD-uri ale firmelor mari prevăd o modalitate mai inteligentă de a realiza operația de compunere între două sau mai multe tabele, metode de indexare noi, potrivite pentru marile cantități de date statice cu care operează sistemele *Data Warehouse*, capacitatea de a detecta şi optimiza interogări de un tip special, posibilitatea de a folosi mai multe procesoare pentru a rezolva o interogare.
- Un sistem *Data Warehouse* are un efect fundamental asupra utilizatorilor. Ei pot manevra mult mai flexibil sistemul, au posibilități elevate pentru interogarea datelor, dar ei trebuie să știe cum să prelucreze şi să vizualizeze datele şi cum să le folosească în procesul de decizie.

Evoluția bazelor de date (32)

- Un efort ce trebuie făcut pentru construirea unui sistem de suport pentru decizii (DSS – *Decision Support System*) constă în procesul de descoperire a informațiilor utile din baza de date.
- Acest proces, numit ***Data Mining*** sau *Knowledge Discovery in Databases* (KDD), procesează mări cantități de date, a căror corelare nu este neapărat evidentă, în vederea descoperirii de tendințe şi tipare.
- Aceste tipare pot fi utilizate pentru stabilirea strategiilor de afaceri sau pentru identificarea unui comportament neobişnuit.
  - De exemplu, o creştere subită a activjtăților din cartea de credit ar putea însemna că respectivul card a fost furat. Instrumentele de explorare a datelor aplică tehnici statistice unor cantități mari de date stocate, pentru a căuta astfel de tipare.

Arhitecturi *multi-user* pentru sisteme de gestiune a bazelor de date (1)

Arhitecturile uzuale care sunt utilizate pentru implementarea sistemelor de gestiune a bazelor de date *multi-user* sunt teleprocesarea, arhitectura fişier-server şi arhitectura client-server.

- **Teleprocesarea** este arhitectura tradiţională, ce cuprinde un calculator cu o singură unitate CPU şi un numar de terminale care sunt incapabile să funcţioneze singure. Terminalele trimit mesaje la programele aplicaţie ale utilizatorilor, care la rândul lor, utilizează serviciile SGBD-ului. Această arhitectură a plasat o greutate teribilă asupra calculatorului central, care pe lângă rularea programelor de aplicaţii şi ale SGBD-ului, mai trebuie să preia şi din munca terminalelor (de exemplu, formatarea datelor pentru afişarea pe ecran).

Arhitecturi *multi-user* pentru sisteme de gestiune a bazelor de date (2)

- **Arhitectura fişier-server**, presupune deja că procesarea este distribuită în rețea (de obicei o rețea locală LAN). Arhitectura cuprinde fișierele cerute de aplicații şi SGBD-ul. Aplicațiile şi funcțiile SGBD sunt executate pe fiecare stație de lucru, solicitând atunci când este nevoie fişiere de pe serverul de fişiere. Dintre dezavantaje se remarcă:
  - existența unui trafic intens pe rețea;
  - necesitatea unei copii complete a SGBD-ului pe fiecare stație de lucru;
  - acelaşi fişier poate fi accesat de mai multe SGBD-uri, ceea ce implică un control complex al integrității, simultaneității şi reconstituirii.

Arhitecturi *multi-user* pentru sisteme de gestiune a bazelor de date (3)

- **Arhitectura client-server** se referă la modul în care interacționează componentele *software* pentru a forma un sistem. Există un proces *client*, care necesită resurse şi un proces *server*, care oferă resurse.
  - În arhitectura client-server, clientul (*front-end*) emite, prin intermediul rețelei locale, o cerere *SQL* care este executată pe server (*back-end*); acesta trimite ca răspuns ansamblul înregistrărilor rezultat. Într-o astfel de interacțiune mașinile sunt eterogene, iar protocoalele de rețea pot fi distincte.
  - În contextul bazelor de date, **clientul**:
    - administrează interfața cu utilizatorul şi logica aplicației;
    - acceptă şi verifică sintaxa intrărilor utilizatorilor;
    - procesează aplicațiile;
    - generează cerințele pentru baza de date şi le trimite serverului;
    - transmite răspunsul înapoi la utilizator.

Arhitecturi *multi-user* pentru sisteme de gestiune a bazelor de date (4)

- În contextul bazelor de date, **serverul**:
  - primeşte şi procesează cerinţele clienţilor pentru baza de date;
  - verifică autorizarea;
  - garantează respectarea constrângerilor de integritate;
  - efectuează procesarea interogare-reactualizare şi trimite clientului răspunsul;
  - realizează optimizarea interogărilor;
  - asigură controlul concurenţei dintre mai mulţi clienţi care se ignoră (mecanisme de blocare);
  - întreţine dicţionarul datelor;
  - oferă acces simultan la baza de date;
  - asigură robusteţea în cazul defecţiunilor;
  - oferă controlul reconstituirii etc.

Arhitecturi *multi-user* pentru sisteme de gestiune a bazelor de date (5)

- Arhitectura tradițională client-server pe „două etaje“ (niveluri) presupune:
  - clientul – responsabil, în primul rând, de prezentarea datelor către client;
  - serverul – responsabil, în primul rând, de furnizarea serviciilor către client.

Arhitecturi *multi-user* pentru sisteme de gestiune a bazelor de date (6)

- Arhitectura client-server pe „trei etaje“ presupune trei niveluri, fiecare fiind rulat, potențial, pe o platformă diferită:
  - nivelul client, format din interfața cu utilizatorul, care este rulat pe calculatorul utilizatorului final;
  - nivelul server de aplicație, ce manevrează logica aplicațiilor și prelucrării datelor, și care poate servi mai mulți clienți (conectarea la celelalte două straturi se face prin rețele locale LAN sau de mare suprafață WAN);
  - nivelul server de baze de date, care se ocupă cu validarea datelor și accesarea bazei de date (stochează date necesare stratului din mijloc).

Arhitecturi *multi-user* pentru sisteme de gestiune a bazelor de date (7)

- Arhitectura se potriveşte natural mediului *Web*, un *browser Web* acționând drept client şi un server *Web* fiind server de aplicație.
- *Middleware* este un strat, evident *software*, între aplicația postului client şi serverul de baze de date, constituit dintr-o interfață de programare a aplicațiilor (API - *Application Programming Interface*) şi un protocol de rețea.
- API descrie tipul de interacțiune dintre o aplicație client şi un server la distanță, *via* un protocol de comunicație şi de formatare a datelor. Scopul existenței interfeței de programare a aplicațiilor este de a oferi o interfață unică mai multor servere de baze de date.