Sisteme de baze de date

Curs 2 – Modelul relațional (cont.)
Proiectarea bazelor de date relaționale

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

Cuprins (Modelul relațional)

JOIN (compunerea, joncţiunea)

- Operatorul de compunere permite regăsirea informaţiei din mai multe relaţii corelate. Compunerea este o operaţie binară care are ca rezultat **o nouă relaţie** în care **fiecare tuplu este o combinaţie a unui tuplu din prima relaţie cu un tuplu din a doua relaţie**.
- Operatorul JOIN este un **operator derivat**, putând fi simulat printr-o combinaţie de **produs cartezian, selecţie şi proiecţie**.
- În general, se construieşte un produs cartezian, se elimină tupluri prin selecţie şi se elimină atribute prin proiecţie.
- După modalităţile în care se face selecţia şi proiecţia, se disting mai multe tipuri de compunere: T**HETA-JOIN, NATURAL-JOIN, SEMI-JOIN, OUTER-JOIN**.

THETA-JOIN

- Operatorul THETA-JOIN combină perechile de tupluri din două relaţii, cu condiţia ca între **valorile atributelor specificate** să existe o anumită legătură, adică **să satisfacă o anumită condiţie specificată explicit în cadrul operaţiei**.
- În cadrul condiţiei operatorului THETA-JOIN se poate folosi orice operator de comparaţie (>, >=, <, <=, <>, =).
- În cazul în care este folosit operatorul de comparaţie =, tipul de compunere se numeşte **EQUI-JOIN**.
- Operatorul THETA-JOIN se reprezintă de obicei cu ajutorul simbolului ⋈ sub care se scrie condiţia, $\mathbf{R} \bowtie_{\text{condiţie}} \mathbf{S}$ sau prin **JOIN(R, S, condiţie)**.

Exemplu

R

A	B	C
x1	y1	1
x2	y2	3
x3	y3	5

S

D	E
2	z1
4	z2

$R \bowtie_{C < D} S$

A	B	C	D	E
x1	y1	1	2	z1
x1	y1	1	4	z2
x2	y2	3	4	z2

THETA-JOIN - cont.

- Următorul exemplu ilustrează realizarea operatorului THETA-JOIN în SQL:

```
SELECT *
FROM R, S
WHERE R.C < S.D;
```

NATURAL-JOIN (Compunerea naturală)

- Compunerea naturală este o operaţie binară comutativă care **combină tupluri din două relaţii, R şi S, cu condiţia ca atributele comune să aibă valori identice.**
- În cazul compunerii naturale atributele specificate trebuie să aibă acelaşi nume.
- Practic diferenţa dintre NATURAL-JOIN şi EQUI-JOIN constă în faptul că în primul caz numele atributelor sunt identice, iar în cel de al doilea caz acestea sunt diferite.
- De obicei, compunerea naturală se notează prin **R ⋈ S** sau **JOIN(R, S)**.

NATURAL-JOIN (Compunerea naturală) - cont.

- Pentru 2 relaţii R şi S, compunerea naturală pe un set de atribute comune X constă în efectuarea succesivă a următoarelor operaţii:
  - Se calculează produsul cartezian R × S.
  - Se selectează din R × S acele tupluri obţinute pentru care valorile atributelor X din tuplul R sunt identice cu valorile atributelor X din tuplul S.
  - Deoarece în relaţia astfel obţinută atributele X apar de două ori (o dată provenind din R şi o dată din S), se elimină una dintre apariţiile acestor atribute.

lucrător

nr_lucrător	cod_secţie	nr_atelier
1	S1	1
2	S1	2
3	S2	1
4	S1	2
5	S1	1

atelier

cod_secţie	nr_atelier	denumire
S1	1	Proiectare
S1	2	Informatica
S2	1	Mecanica
S2	2	Electrotehnica

lucrător ⋈ atelier

nr_lucrător	cod_secţie	nr_atelier	denumire
1	S1	1	Proiectare
2	S1	2	Informatica
3	S2	1	Mecanica
4	S1	2	Informatica
5	S1	1	Proiectare

NATURAL-JOIN (Compunerea naturală) - cont.

- În exemplul de mai sus, {`cod_secţie, nr_atelier`} este cheie primară în tabelul `atelier` şi cheie străină în tabelul `lucrător`.
- Următorul exemplu ilustrează realizarea compunerii naturale în SQL:

```
SELECT lucrător.nr_lucrător, lucrător.cod_secţie,
lucrător.nr_atelier, atelier.denumire
FROM lucrător, atelier
WHERE lucrător.cod_secţie = atelier.cod_secţie
AND lucrător.nr_atelier = atelier.nr_atelier;
```

SEMI-JOIN (semi-compunerea)

- Operaţia de semi-compunere aplicată asupra a două relaţii R şi S generează **o relaţie care conţine toate tuplurile din R corelate cu oricare din tuplurile din S**.
- Operaţia nu este comutativă.
- Se notează de obicei prin **SEMI-JOIN(R, S)**.

Exemplu

R

A	B	C
x1	y1	z1
x2	y1	z1
x3	y2	z1
x4	y2	z2

S

B	C	D
y1	z1	u1
y2	z2	u2
y2	z2	u3

SEMIJOIN(R,S)

A	B	C
x1	y1	Z1
x2	y1	z1
x4	y2	z2

SEMI-JOIN (semi-compunerea) - cont.

- Următorul exemplu ilustrează realizarea semi-compunerii în SQL:

```
SELECT DISTINCT R.A, R.B, R.C
FROM R, S
WHERE R.B = S.B
AND R.C = S.C;
```

OUTER-JOIN (Compunerea externă)

- Operaţia de compunere externă este o extindere a compunerii naturale.
- În cazul aplicării operatorului NATURAL-JOIN se pot pierde tupluri atunci când există un tuplu într-una din relaţii care nu este corelat cu nici un tuplu din cealaltă relaţie.
- Operatorul OUTER-JOIN elimină acest inconvenient. Practic, la aplicarea operatorului OUTER-JOIN, se realizează **compunerea naturală a celor două relaţii, la care se adaugă tuplurile din S care nu sunt conţinute în compunere, completate cu valori Null pentru atributele rămase din R**.

OUTER-JOIN (Compunerea externă)

- Operatorul se notează cu **OUTERJOIN(R, S)**.
- Există şi alte variante ale acestui operator, de exemplu o altă variantă adaugă la tuplurile obţinute din compunerea naturală a lui R şi S atât tuplurile din R care nu sunt conţinute în compunere cât şi tuplurile din S care nu sunt conţinute în compunere, completând restul cu Null.
  În mod evident, această variantă a operatorului se poate obţine cu uşurinţă din varianta prezentată de noi.
- Operatorul OUTER-JOIN, în varianta prezentată de noi, este ne-comutativ.

Exemplu

`student`

nr_stud	nume	prenume	cod_facult
1	Popescu	Ion	F1
2	Ionescu	Vasile	F1
3	Ionescu	Viorel	F2
4	Costache	Ion	F2
5	Matache	Mihai	F1

`facultate`

cod_facult	nume_facult	localitate
F1	Matematica	Bucureşti
F2	Fizica	Bucureşti
F3	Informatica	Piteşti
F4	Mecanica	Ploieşti

```
OUTERJOIN(student,facultate)
```

nr_stud	nume	prenume	cod_facult		
1	Popescu	Ion	F1	Matematica	Bucureşti
2	Ionescu	Vasile	F1	Matematica	Bucureşti
3	Ionescu	Viorel	F2	Fizica	Bucureşti
4	Costache	Ion	F2	Fizica	Bucureşti
5	Matache	Mihai	F1	Matematica	Bucureşti
Null	Null	Null	F3	Informatica	Piteşti
Null	Null	Null	F4	Mecanica	Ploieşti

OUTER-JOIN (Compunerea externă) - cont.

- În versiunea SQL folosită de Oracle, operatorul OUTER JOIN este specificat prin sufixul (+) adăugat la câmpul după care se face compunerea, corespunzător tuplului ale cărui atribute pot fi completate cu Null:

```
SELECT student.nr_stud, student.nume, student.prenume,
       facultate.cod_facult, facultate.nume_facult,
       facultate.localitate
FROM student, facultate
WHERE student.cod_facult (+) = facultate.cod_facult;
```

IV. Tabele, rânduri, coloane

- Modelul relaţional este bazat pe matematica relaţională.
- Preluarea modelului relaţional de către economie a implicat o transformare a terminologiei relaţionale într-una care poate fi uşor înţeleasă de cei fără o pregătire specială în domeniu.
- Pentru cei care nu sunt experţi în procesarea datelor, **relaţiile devin tabele**, **tuplurile devin rânduri** şi **atributele devin coloane**.
  Acesta este şi terminologia pe care o vom folosi în continuare.

V. Sisteme de Gestiune a Bazelor de Date Relaţionale (SGBDR)

- În principiu, un sistem de gestiune a bazelor de date relaţionale (SGBDR) este un SGBD care utilizează drept concepţie de organizare a datelor modelul relaţional.
- Evident definiţia este mult prea generală pentru a putea fi folosită în practică, deoarece modul de implementare a modelului relaţional diferă de la un producător la altul.
- În 1985, Codd a publicat **un set de 13 reguli în raport cu care un SGBD poate fi apreciat ca relaţional**.

V. SGBDR - cont.

- **Nici un SGBD comercializat în prezent nu satisface în totalitate regulile lui Codd**, dar aceasta nu împiedică etichetarea acestora drept relaţionale.
- Regulile lui Codd nu trebuie folosite pentru a aprecia dacă un sistem este sau nu relaţional, ci măsura în care acesta este relaţional, adică nr. regulilor lui Codd respectate de către acesta.

Regulile lui Codd

- **Regula 1. Regula reprezentării logice a datelor:** Într-o BD relaţională, toate datele sunt reprezentate la nivel logic într-un singur mod, şi anume sub formă de valori atomice în tabele.

- Deci toate datele trebuie memorate sub formă de tabele, iar valoarea corespunzătoare intersecţiei dintre un rând şi o coloană trebuie să fie atomică, adică să nu mai poată fi descompusă din punct de vedere logic.

Regulile lui Codd - cont.

- Un exemplu de încălcare a acestei reguli este stocarea ca o singură coloană a unui cod al automobilului, obţinut prin concatenarea mai multor coduri, reprezentând marca automobilului, culoarea, fabrica unde este produs, seria şi numărul de fabricaţie, etc.
- Uneori această regulă este încălcată în practică, dar acest lucru este de cele mai multe ori semnul unui design de calitate slabă, creând probleme de integritate a BD.
- **Această regulă este cea mai importantă dintre cele definite de Codd**, iar un SGBD care nu respectă această regulă nu poate fi în nici un caz considerat relaţional.

Regulile lui Codd - cont.

- **Regula 2. Regula accesului la date:** Toate datele individuale din tabele trebuie să fie accesibile prin furnizarea numelui tabelului, numelui coloanei şi valorii cheii primare.

- Conform modelului relaţional, într-un tabel nu pot exista rânduri identice, iar fiecare rând poate fi identificat prin valoarea cheii primare.
  În consecinţă, orice dată individuală poate fi identificată folosind numele tabelului, al coloanei şi valoarea cheii primare.

Regulile lui Codd - cont.

- **Oracle nu respectă această regulă** deoarece permite existenţa a mai multor rânduri identice în acelaşi tabel.
- Totuşi, acest lucru poate fi evitat, de exemplu prin definirea unei chei primare, care elimină implicit şi posibilitatea existenţei rândurilor identice.
- Pe de altă parte, această regulă este încălcată în Oracle şi de existenţa identificatorului de rând, ROWID, care poate fi folosit pentru accesarea rândului respectiv.

Regulile lui Codd - cont.

- **Regula 3. Regula reprezentării valorilor necunoscute:** Un sistem relaţional trebuie să permită declararea şi manipularea sistematică a valorilor Null, cu semnificaţia unor valori necunoscute sau inaplicabile.
- Această regulă, implică, de exemplu, că un SGBDR trebuie să facă diferenţa între valoarea numerică 0 şi Null sau între şirul de caractere „spaţiu” şi valoarea Null.
- Valoarea Null trebuie să reprezinte absenţa informaţiei respective şi are un rol important în implementarea restricţiilor de integritate structurală (integritatea entităţii şi integritatea referirii).
- **Oracle respectă această regulă**, limbajul SQL permiţând declararea şi manipularea valorilor Null.

Regulile lui Codd - cont.

- **Regula 4. Regula dicţionarului de date:** Descrierea BD (i.e. dicţionarul de date) trebuie să fie reprezentată la nivel logic tot sub formă de tabele, astfel încât asupra acesteia să se poată aplica aceleaşi operaţii ca şi asupra datelor propriu-zise.
- Cu alte cuvinte, dicţionarul de date trebuie să fie organizat la nivel logic şi accesat la fel ca orice tabel din baza de date.
- **Această regulă este respectată de către Oracle**, dicţionarul de date constând din tabele şi tabele virtuale (vederi) care pot fi interogate la fel ca oricare alte tabele sau vederi, folosind comanda SELECT.

Regulile lui Codd - cont.

- **Regula 5. Regula limbajului de acces:** Într-un sistem relaţional trebuie să existe cel puţin un limbaj de accesare a datelor, care să asigure următoarele operaţii:
  - definirea tabelelor de bază şi a tabelelor virtuale (vederilor),
  - manipularea şi interogarea datelor (atât interactiv cât şi prin program),
  - definirea restricţiilor de integritate,
  - autorizarea accesului la date,
  - delimitarea tranzacţiilor.

Regulile lui Codd - cont.

- **Limbajul SQL folosit de către Oracle permite**
  - definirea tabelelor (comenzile CREATE TABLE, ALTER TABLE, DROP TABLE), a vederilor (comenzile CREATE VIEW, ALTER VIEW, DROP VIEW),
  - manipularea (comenzile INSERT, UPDATE, DELETE) şi interogarea acestora (comanda SELECT),
  - definirea restricţiilor de integritate (clauza CONSTRAINT folosită la definirea tabelelor),
  - autorizarea accesului la date (printr-un set de privilegii de sistem şi la nivel de obiect),
  - delimitarea tranzacţiilor (operaţiile COMMIT şi ROLLBACK).

Regulile lui Codd - cont.

- **Regula 6. Regula de actualizare a tabelelor virtuale (vederilor):** Un SGBD trebuie să poată determina dacă o vedere poate să fie actualizată sau nu.

- **Un tabel virtual (vedere)** este un tabel logic, în sensul că el organizează datele sub forma unor rânduri şi coloane, ca orice alt tabel, dar în schimb el nu stochează datele, fiind construit pe baza unor interogări asupra unuia sau mai multor tabele de bază.

Regulile lui Codd - cont.

- De exemplu, să considerăm tabelul : `salariu(cod_salariat, salariu_brut, zile_totale, zile_lucrate)`.
- Pe baza acestui tabel se poate defini vederea `salariu_r(cod_salariat, salariu_brut, salariu_realizat)` unde `salariu_realizat` este definit ca `salariu_realizat = salariu_brut*zile_totale/zile_lucrate`
- Să presupunem că se doreşte actualizarea coloanei `salariu_brut` din vedere.

Regulile lui Codd - cont.

- Acest lucru este posibil, datorită faptului că actualizarea se propagă înapoi la coloana din tabelul de bază, producându-se şi actualizarea acesteia.
- Pe de altă parte, nu este posibilă actualizarea coloanei `salariu_realizat`, datorită faptului că schimbarea valorii acesteia s-ar putea produce datorită schimbării valorilor mai multor coloane (`salariu_brut, zile_totale sau zile_lucrate`), SGBD-ul neştiind care din aceste coloane trebuie actualizată în tabelul de bază.
- **Oracle respectă această regulă**, existând un set de reguli care determină dacă o coloană a unei vederi poate sau nu să fie actualizată.

Regulile lui Codd - cont.

- **Regula 7. Regula manipulării datelor:** Un sistem relaţional trebuie să ofere posibilitatea procesării tabelelor (de bază sau virtuale) nu numai în operaţiile de interogare a datelor cât şi în cele de inserare, actualizare şi ştergere.

- Aceasta înseamnă că operaţiile de manipulare a datelor (inserare, actualizare şi ştergere) trebuie să se poată efectua asupra oricărei mulţimi de rânduri dintr-un tabel, pornind de la întregul tabel şi terminând cu un singur rând sau cu nici unul.

Regulile lui Codd - cont.

- Deci, un SGBD relaţional nu obligă utilizatorul să caute într-un tabel rând cu rând pentru a regăsi, modifica sau şterge informaţia dorită, deoarece operaţiile prin care se manipulează conţinutul BD lucrează la nivel de mulţime de rânduri.

- **Limbajul SQL asigură această facilitate** prin instrucţiunile: INSERT cu subinterogare, UPDATE şi DELETE.

Regulile lui Codd - cont.

- **Regula 8. Regula independenţei fizice a datelor:** Programele de aplicaţie nu trebuie să depindă de modul de stocare şi accesare fizică a datelor.

- Deci un SGBD relaţional trebuie să separe complet aspectele de ordin fizic ale datelor (modul de stocare şi modul de acces la date) de cele de ordin logic.

Regulile lui Codd - cont.

- De exemplu, dacă un fişier care conţine un tabel de date este mutat pe o altă unitate de disc sau îi este schimbat numele, aceasta nu trebuie să aibă vreun efect asupra aplicaţiilor care folosesc acel tabel, utilizatorilor fiindu-le transparentă această schimbare.

- **În mare, Oracle respectă această regulă**, deşi stocarea fizică a datelor trebuie luată în consideraţie la proiectarea bazei de date.

Regulile lui Codd - cont.

- **Regula 9. Regula independenţei logice a datelor:** Programele de aplicaţie nu trebuie să fie afectate de nici o restructurare logică a tabelelor BD care conservă datele.

- Deci orice modificare efectuată asupra unui tabel care conservă datele din acesta (de exemplu, dacă un tabel trebuie divizat în două, din raţiuni de creştere a performanţelor) nu trebuie să afecteze funcţionarea programelor de aplicaţie.

Regulile lui Codd - cont.

- **Această regulă este respectată de către Oracle** prin posibilitatea definirii vederilor: dacă un tabel este divizat în 2, atunci se poate crea o vedere care alătură cele 2 tabele, astfel încât această împărţire nu va avea nici un efect asupra aplicaţiei.

Regulile lui Codd - cont.

- **Regula 10. Regula independenţei datelor din punctul de vedere al integrităţii:** Regulile de integritate a BD trebuie să fie definite în limbajul utilizat de sistem pentru definirea datelor şi nu în cadrul aplicaţiilor individuale; în plus, aceste reguli de integritate trebuie stocate în dicţionarul de date.

- Cu alte cuvinte, restricţiile de integritate trebuie impuse la definirea tabelelor BD şi nu în cadrul aplicaţiilor care folosesc aceste tabele.

Regulile lui Codd - cont.

- **În general, Oracle respectă această regulă**, la definirea tabelelor (în cadrul comenzii CREATE TABLE) putându-se defini atât restricţiile de integritate structurală (NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY) cât şi unele restricţii de comportament (CHECK).
  Informaţii despre aceste restricţii sunt stocate în dicţionarul bazei de date.

Regulile lui Codd - cont.

- **Regula 11. Regula independenţei datelor din punctul de vedere al distribuirii:** Programele de aplicaţie nu trebuie să fie afectate de distribuirea pe mai multe calculatoare a BD.

- Cu alte cuvinte, BD trebuie să meargă corect indiferent dacă se găseşte pe un singur calculator sau este distribuită în mai multe noduri ale unei reţele.
- Această regulă este o extensie a regulii 8, privind independenţa programelor de aplicaţie faţă de modul de stocare fizică a datelor.

Regulile lui Codd - cont.

- **Această regulă este în general respectată de Oracle**, existând totuşi restricţii privind accesarea unor obiecte aflate în alt nod al reţelei.
- În plus, în Oracle există posibilitatea replicării locale a tabelelor aflate în alte noduri ale reţelei, evitându-se astfel transmiterea în mod repetat a informaţiilor prin reţea.

Regulile lui Codd - cont.

- **Regula 12. Regula privind prelucrarea datelor de către un limbaj de nivel inferior:** Orice limbaj nerelaţional folosit pentru accesarea datelor trebuie să respecte aceleaşi condiţii de integritate ca şi limbajul relaţional de acces.

- De exemplu, dacă sistemul posedă un limbaj de nivel inferior, prin care se accesează datele la nivel de rând şi nu, potrivit sistemului relaţional, la nivelul mulţimilor de rânduri, acest limbaj nu poate fi folosit pentru evitarea restricţiilor de integritate (integritatea entităţii, integritatea referenţială, restricţii de comportament) pe care trebuie să le respecte limbajul procedural de acces la date.

Regulile lui Codd - cont.

- **Această regulă este respectată de către Oracle** prin faptul că singurul limbaj de accesare a datelor este SQL, care este un limbaj relaţional.

- Dacă un SGBD îndeplineşte principiile sistemului relaţional (foloseşte ca structuri de date tabele conforme cu modelul relaţional, suportă cele două reguli de integritate structurală şi operaţiile relaţionale) şi respectă aceste 12 reguli, atunci el poate fi numit relaţional.

- **Codd rezumă aceste lucruri prin regula zero**:

Regulile lui Codd - cont.

- **Regula 0. Regula de bază:** Un SGBD Relaţional trebuie să fie capabil să gestioneze BD exclusiv pe baza caracteristicilor sale relaţionale.

- Aceasta înseamnă că sistemul trebuie să-şi îndeplinească toate funcţiile prin manipulări în care unitatea de procesare să fie tabelul (mulţimea de rânduri), asupra căruia să se efectueze operaţiile specifice modelului relaţional.
- **Regula 0 nu este respectată în totalitate de nici un SGBD existent pe piaţă**, **inclusiv Oracle**, implementarea acestora folosind atât caracteristici relaţionale cât şi nerelaţionale.

Regulile lui Codd - cont.

- Se obişnuieşte ca, în conformitate cu tipul de cerinţe pe care le exprimă, regulile să fie grupate în 5 categorii, şi anume:
  1. reguli de bază: Regula 0 şi Regula 12;
  2. reguli structurale: Regula 1 şi Regula 6;
  3. reguli privind integritatea datelor: Regula 3 şi Regula 10;
  4. reguli privind manipularea datelor: Regula 2, Regula 4, Regula 5 şi Regula 7;
  5. reguli privind independenţa datelor: Regula 8, Regula 9 şi Regula 11.
- Trebuie spus că **nici unul dintre SGBD-urile existente în prezent nu satisface în totalitate toate cele 13 reguli ale lui Codd**.

Regulile lui Codd - cont.

- De aceea, în practică nu sunt utilizate regulile lui Codd, fiind formulate în schimb **un set de cerinţe minimale pe care trebuie să le satisfacă un sistem SGBD pentru a putea fi considerat relaţional**.

Regulile lui Codd - cont.

- **Un SGBD este denumit minimal relaţional**, dacă satisface următoarele condiţii:
  - Toate datele din cadrul BD sunt reprezentate prin valori în tabele.
  - Nu există pointeri între tabele observabile de către utilizator.
  - Sistemul suportă operatorii relaţionali de proiecţie, selecţie şi compunere naturală, fără limitări impuse de considerente interne.

Regulile lui Codd - cont.

- **Un SGBD este denumit complet relaţional** dacă este minimal relaţional şi satisface în plus următoarele condiţii:
  - Sistemul suportă toate operaţiile de bază ale algebrei relaţionale, fără limitări impuse de considerente interne.
  - Sistemul suportă restricţiile de integritate de bază ale modelului relaţional (integritatea entităţii şi integritatea referenţială).

- **SGDB-ul Oracle este complet relaţional şi chiar se apropie destul de mult de un SGBD relaţional ideal, definit prin regulile lui Codd.**

Cuprins (Proiectarea bazelor de date)

1. Crearea schemei conceptuale
    a. Modelul entitate-legătură (entitate-relaţie)
2. Crearea design-ului logic al bazei de date
3. Crearea design-ului fizic al bazei de date

Proiectarea bazelor de date

- După mai bine de 2 decenii de folosire a modelului relaţional, **proiectarea (designul) BD rămâne încă mai degrabă artă decât ştiinţă.**
- Au fost sugerate un număr de metode, dar până în prezent nici una nu este dominantă.
- Pe de altă parte proiectarea bazelor de date trebuie să fie bazată pe consideraţii practice care stau la baza oricărei activităţi de procesare a datelor.
- Pentru a crea un design adecvat este necesară o cunoaştere aprofundată a funcţionării antreprizei, a modului în care aceasta foloseşte datele şi a sistemului de management al bazelor de date folosit.

Proiectarea bazelor de date - cont.

- Metodele curente de proiectare a BD sunt în general divizate în 3 etape separate:
  - crearea schemei conceptuale,
  - crearea design-ului logic al bazei de date şi
  - crearea design-ului fizic al bazei de date.

Proiectarea bazelor de date - cont.

- **Crearea schemei conceptuale**.
  - Aceasta este un design de nivel înalt (incluzând relaţiile dintre datele întregului sistem), care descrie datele şi relaţiile necesare pentru execuţia operaţiilor necesare, fiind independent de orice model de baze de date.
  - Designul de la acest nivel este foarte general, se realizează într-o perioadă scurtă de timp şi prezintă modul în care grupările de date sunt integrate în sistemul de ansamblu.

Proiectarea bazelor de date - cont.

- **Crearea design-ului logic al bazei de date**.
  - În această fază, schema conceptuală este transformată în structuri specifice unui anumit sistem de management al bazei de date.
  - La acest nivel designul este rafinat, sunt definite elemente de date specifice şi sunt grupate în înregistrări.
  - În cazul modelului relaţional, la sfârşitul acestei etape vom avea un număr de tabele care vor permite stocarea şi manipularea corectă a tuturor datelor necesare sistemului.

Proiectarea bazelor de date - cont.

- **Crearea design-ului fizic al bazei de date**.
  - În această etapă designul logic este transformat într- o structură fizică eficientă.

1. Crearea schemei conceptuale

- Procesul de design al schemei conceptuale începe prin determinarea datelor necesare activităţilor din antrepriză.
- Este creată o echipă de design a schemei conceptuale care se ocupă cu determinarea datelor necesare, eventual prin folosirea de interviuri cu managerii antreprizei.
- După ce echipa proiectează datele, ea le revizuieşte şi le organizează.
- **Diagrama conceptuală se construiește din diagrama E/R prin adăugarea tabelelor asociative şi prin marcarea cheilor externe.**

1.a. Modelul entitate-legătură (entitate-relaţie)

- Una dintre tehnicile folosite pentru organizarea rezultatelor din etapa de colectare a datelor este modelul entitate-legătură, care **împarte elementele unui sistem real în 2 categorii** şi anume **entităţi şi legături (relaţii) între aceste entităţi**.
- Principalele concepte folosite în acest model sunt cele de entitate, relaţie (legătură) şi atribut.
- **Notă**: Nu trebuie confundat conceptul de relaţie în sensul de legătură sau asociere, care intervine în modelul entitate-legătură cu cel definit în cursul 1.

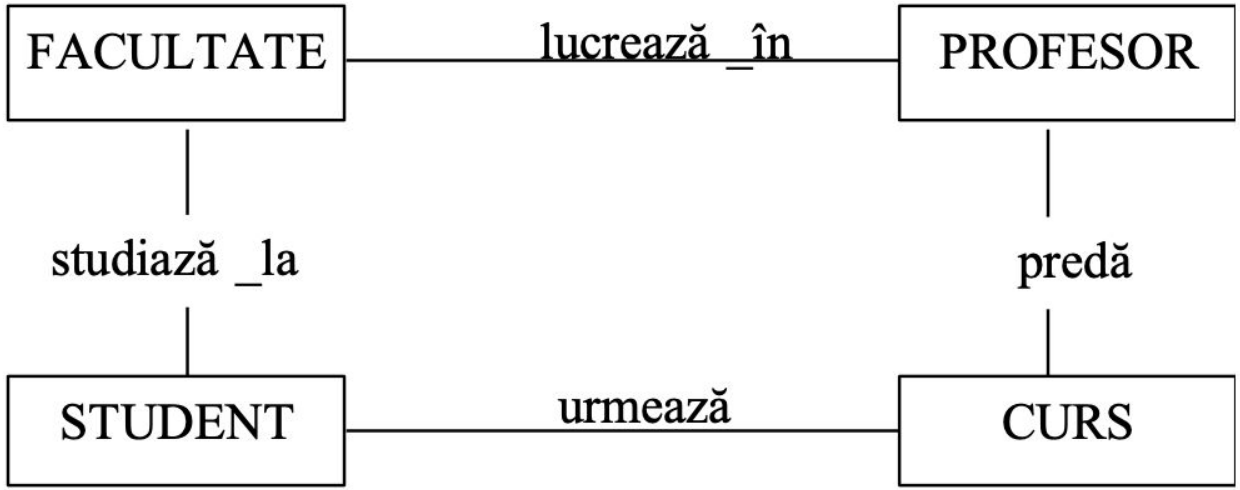
Entitate

- Este un obiect de interes şi pentru care trebuie să existe date înregistrate.
- Poate fi atât un obiect tangibil – precum **persoane, locuri sau lucruri** - cât şi abstracte – precum **comenzi, conturi bancare**, etc.

Entitate - cont.

- De exemplu, să considerăm o universitate formată din mai multe facultăţi; în fiecare **facultate** studiază mai mulţi **studenţi** şi predau mai mulţi **profesori**. Fiecare **student** urmează mai multe **cursuri**, după cum un **profesor** poate preda unul sau mai multe cursuri.
- Elementele semnificative ale acestui sistem sunt: facultate, student, profesor şi curs; acestea sunt entităţile sistemului.
- Ele sunt reprezentate în figura de pe slide-ul următor împreună cu relaţiile dintre ele.
- Remarcaţi că entităţile se reprezintă prin dreptunghiuri, iar relaţiile dintre ele prin arce neorientate.

Entitate - cont.



Entitate - cont.

- Ideile de bază pentru identificarea şi reprezentarea entităţilor sunt următoarele:
  - Fiecare entitate este denumită în mod unic; nu pot exista 2 entităţi cu acelaşi nume sau o entitate cu 2 nume diferite.
  - Entităţile sunt reprezentate întotdeauna prin substantive, dar nu orice substantiv folosit în descrierea sistemului este o entitate a acestuia.
  - Entităţile sistemului sunt doar acele substantive care au o semnificaţie deosebită în descrierea sistemului.

Entitate - cont.

De exemplu, chiar dacă suntem interesaţi de nr. de ore de predare efectuate de un profesor pe săptămână, aceasta nu înseamnă că vom crea o entitate pentru aceasta. De fapt, vom vedea în continuare că nr. de ore predate va fi un atribut al entităţii PROFESOR.

- De asemenea, pentru fiecare entitate trebuie să se dea o descriere detaliată, de exemplu, putem spune că un PROFESOR este un cadru didactic angajat al universităţii pe o perioadă nedeterminată, din această categorie făcând parte atât profesorii permanenţi cât şi cei asociaţi, dar fiind excluşi cei care predau la universitate numai o perioadă limitată.

Relație (legătură)

- Entităţile pot forma relaţii între ele. O relaţie este o asociere nedirecţionată între 2 entităţi. Ea exprimă **un raport care există între entităţile respective**.

- De exemplu, „lucrează_în” este o relaţie între entităţile PROFESOR şi FACULTATE, iar „predă” este o relaţie între entităţile PROFESOR şi CURS.

- Principalele idei pentru identificarea şi reprezentarea relaţiilor sunt următoarele:
  - Relaţiile sunt reprezentate prin verbe, dar nu orice verb este o relaţie.

Relație (legătură) - cont.

- Între 2 entităţi poate exista mai mult decât o singură relaţie.
  De exemplu, dacă luăm în vedere că fiecare facultate este condusă de un decan şi că acesta este ales din rândurile profesorilor, atunci între entităţile PROFESOR şi FACULTATE va mai exista o relaţie numită „conduce”.

- Pot exista relaţii cu acelaşi nume, dar relaţiile care asociază aceleaşi entităţi trebuie să poarte nume diferite.

Relație (legătură) - cont.

- **Cardinalitatea unei relaţii** indică nr. maxim de instanţe din fiecare entitate care poate participa la relaţie.
- Cu alte cuvinte, cardinalitatea unei relaţii reprezintă răspunsul la întrebări de genul: Câţi studenţi pot studia la o facultate? Mulţi.
  Dar la câte facultăţi poate studia un student? La cel mult una.
  Deci cardinalitatea relaţiei „studiază_la” este de mulţi-la-unu.

- Cardinalitatea unei relaţii poate fi de trei feluri: **mulţi-la-unu**, **unu-la-unu** sau **mulţi-la-mulţi**.

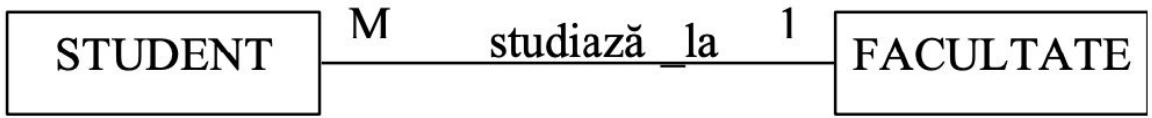
Relație (legătură) - cont.

- **mulţi-la-unu (many-to-one, N:1)**: Relaţia dintre entităţile A şi B este de tipul mulţi-la-unu dacă fiecărei instanţe din A îi poate fi asociată cel mult o singură instanţă din B şi fiecărei instanţe din B îi pot fi asociate mai multe instanţe din A.

- De exemplu, relaţiile „lucrează_în” dintre PROFESOR şi FACULTATE şi „studiază_la” dintre STUDENT şi FACULTATE sunt de tipul N:1.

Relație (legătură) - cont.

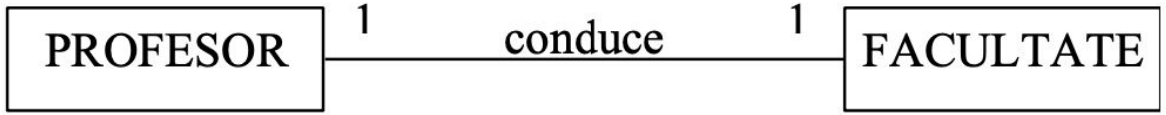
➢ O relaţie mulţi-la-unu se reprezintă în modul următor:



Relație (legătură) - cont.

- **unu-la-unu (one-to-one, 1:1)**: Relaţia dintre entităţile A şi B este de tipul unu-la-unu dacă fiecărei instanţe din A îi poate fi asociată cel mult o singură instanţă din B şi fiecărei instanţe din B îi poate fi asociată cel mult o singură instanţă din A.

- De exemplu, relaţia „conduce” dintre PROFESOR şi FACULTATE este o relaţie 1:1.



Relație (legătură) - cont.

- **mulţi-la-mulţi (many-to-many, N:M)**: Relaţia dintre entităţile A şi B este de tipul mulţi-la-mulţi dacă fiecărei instanţe din A îi pot fi asociate mai multe instanţe din B şi fiecărei instanţe din B îi pot fi asociate mai multe instanţe din A.

- De exemplu, relaţiile „predă” dintre PROFESOR şi CURS şi „urmează” dintre STUDENT şi CURS sunt de tipul N:M. O relaţie N:M se reprezintă în modul următor:

Relație (legătură) - cont.

- Valorile discutate până acum (N:1, 1:1, N:M) reprezintă **cardinalitatea maximă** a unei relaţii. Pe de altă parte, o relaţie este caracterizată şi de o **cardinalitate minimă**, care indică obligativitatea participării entităţilor la relaţie.
  Cu alte cuvinte, aceasta furnizează răspunsul la întrebări de genul:

  Câţi studenţi trebuie să studieze la o facultate? Zero.
  Dar la câte facultăţi trebuie să studieze un student? Cel puţin una.

  Deci cardinalitatea minimă a relaţiei „studiază_la” dintre STUDENT şi FACULTATE este de 0:1.

Relație (legătură) - cont.

- În mod similar, relaţia „predă” dintre PROFESOR şi CURS are cardinalitatea minimă 0:0 (un profesor trebuie să predea zero cursuri şi un curs trebuie să fie predat de zero profesori – de exemplu dacă cursul este nou şi nu s-a stabilit încă titularul de curs).
  Deci **cardinalitatea minimă** a unei relaţii poate avea valorile **0:0, 0:1 şi 1:1**.
- Dacă participarea unei entităţi la o relaţie este obligatorie (cardinalitatea minimă respectivă este 1) se mai spune şi că **participarea** acesteia la relaţie este **totală**.
- În caz contrar (cardinalitatea minimă respectivă este 0), participarea entităţii la relaţie se numeşte **parţială**.

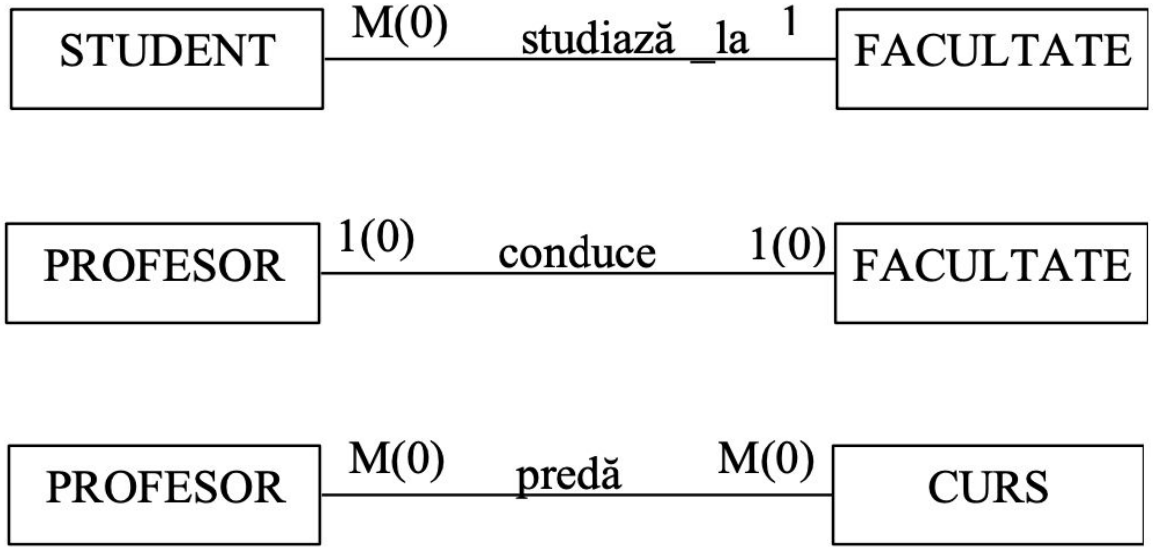
Relație (legătură) - cont.

- De exemplu participarea entităţii STUDENT la relaţia „studiază_la” este parţială, pe când participarea entităţii FACULTATE la aceeaşi relaţie este totală.

- În cadrul reprezentării grafice, **cardinalitatea maximă** a unei relaţii se va indica **fără paranteze**, în timp ce c**ardinalitatea minimă**, dacă este diferită de cea maximă, se va scrie **în paranteze**, vezi figurile de pe slide-ul următor.

- De multe ori, cardinalitatea minimă nu este indicată în diagrama entitate-legătură, pe când cardinalitatea maximă trebuie indicată întotdeauna, ea fiind esenţială.

Relație (legătură) - cont.

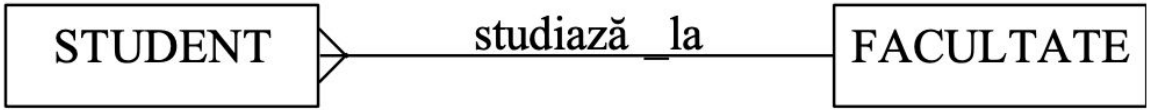


Relație (legătură) - cont.

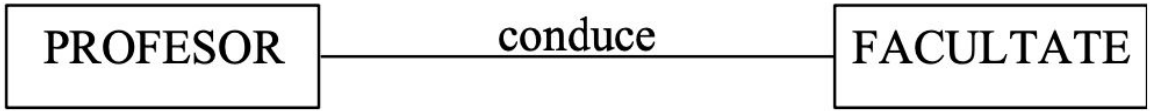
- Un alt mod de a reprezenta relaţiile, indicând doar cardinalitatea lor maximă este următorul:

Relație (legătură) - cont.

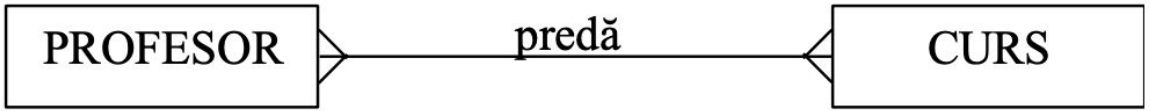
- relaţii mulţi-la-unu



- relaţii unu-la-unu



- relaţii mulţi-la-mulţi



Atribut

- Un atribut este **o caracteristică a unei entităţi sau a unei relaţii**.
  Fiecare entitate are un anumit număr de atribute despre care sunt înregistrate date.

- De exemplu, numele, prenumele, vârsta şi numărul de ore predate sunt atribute ale entităţii PROFESOR.

- Fiecare atribut poate lua o valoare care furnizează informaţii despre entitatea respectivă.

Atribut

- Exemple de valori de atribute sunt „Ionescu” pentru nume, „Mihai” pentru prenume etc.

- Pe de altă parte şi relaţiile pot avea atribute.
  De exemplu, relaţia „urmează” dintre STUDENT şi CURS poate avea ca atribute nota obţinută la examen şi nota obţinută la restanţă - pentru cei care nu au promovat examenul - iar relaţia „lucrează_în” dintre PROFESOR şi FACULTATE poate avea ca atribut data angajării.

Atribut - cont.

- Principalele idei pentru identificarea şi reprezentarea atributelor sunt următoarele:
  - Numele unui atribut este unic în cadrul unei entităţi sau al unei relaţii.
  - Atributele sunt întotdeauna substantive, dar nu orice substantiv este un atribut.
  - Pentru fiecare atribut, trebuie furnizată o descriere, împreună cu domeniul de valori (întreg, şir de caractere, dată, etc.).

Atribut - cont.

- Alegerea atributelor trebuie făcută în aşa fel încât să se evite aşa-numitele atribute indirecte.

- Un **atribut indirect** al unei entităţi sau relaţii este un atribut care nu aparţine în mod real acelei entităţi sau relaţii, fiind o caracteristică a unui alt obiect al sistemului.

- De exemplu, numele facultăţii este un atribut indirect al entităţii STUDENT, el descriind de fapt o proprietate a entităţii FACULTATE. De aceea, el va trebui redistribuit acestei entităţi.

Modelul entitate-legătură şi modelul relaţional

- Modelul entitate-legătură poate fi transformat în mod natural într-o bază de date relaţională. Fără a intra deocamdată în amănuntele acestei transformări, enunţăm în continuare principalele idei ale acestei transformări:

  - O entitate devine un tabel.
  - Un atribut al unei entităţi devine o coloană a tabelului respectiv.
  - O relaţie va fi reprezentată fie printr-un tabel special, fie printr-o cheie străină într-unul dintre cele două tabele entitate, care face referire la cheia primară a celuilalt tabel entitate.

Chei primare. Chei naturale şi chei artificiale

- În concordanţă cu terminologia folosită în cursul 1, o cheie a unei entităţi va fi un atribut sau un set de atribute care identifică în mod unic o instanţă a acelei entităţi.
- Cu alte cuvinte, o cheie face distincţie între oricare două rânduri diferite ale tabelului provenit din entitatea respectivă.
- De exemplu, putem presupune că fiecare student va fi identificat în cadrul universităţii printr-un cod unic; atunci codul studentului este o cheie a entităţii STUDENT.

Chei primare. Chei naturale şi chei artificiale

- Pe de altă parte, numele studentului nu poate fi cheie a acestei entităţi deoarece pot exista mai mulţi studenţi cu acelaşi nume.

- Dacă însă presupunem că nu pot exista studenţi cu acelaşi nume, prenume şi dată de naştere atunci combinaţia acestor atribute este la rândul ei cheie a entităţii STUDENT.

Chei primare. Chei naturale şi chei artificiale

- Există 2 tipuri de chei: naturale şi artificiale.
- O **cheie naturală** este constituită dintr-un atribut sau o combinaţie de atribute cu semnificaţie reală pentru entitatea în cauză.
  De exemplu, combinaţia nume, prenume, dată de naştere este o cheie naturală a entităţii STUDENT.
- O **cheie artificială** este un atribut al unei entităţi care nu are semnificaţie reală pentru entitatea în cauză, fiind folosită doar pentru a face distincţie între instanţele entităţii.
  De exemplu, codul studentului este o cheie artificială a entităţii STUDENT.

Chei primare. Chei naturale şi chei artificiale

- Una dintre cheile entităţii va fi declarată cheie primară (notăm cu CP sau PK).
- Deci, în principiu, oricare dintre cele 2 chei ale entităţii STUDENT poate fi declarată CP.
- Pe de altă parte însă, este preferată folosirea CP artificiale, excepţie făcând cazul când CP respectivă nu va fi stocată în alte tabele ca şi cheie străină.

Chei primare. Chei naturale şi chei artificiale

- Principalele avantaje ale CP artificiale faţă de cele naturale sunt următoarele:
  - **Stabilitatea**. Valoarea unei chei artificiale rămâne aceeaşi pe parcursul funcţionării sistemului, în timp ce valoarea unei chei naturale poate fi în general modificată, această modificare atrăgând, la rândul ei, schimbarea cheilor străine care fac referire la ea.
    De exemplu, numele unei studente se poate schimba prin căsătorie; dacă se consideră combinaţia nume, prenume şi data naşterii ca fiind CP a entităţii STUDENT, atunci orice schimbare a numelui va impune modificarea valorilor cheilor străine corespunzătoare.

Chei primare. Chei naturale şi chei artificiale

- Ca o regulă generală, valoarea CP a unui tabel nu trebuie să poată fi modificată, aceasta creând probleme privind păstrarea integrităţii datelor - cu alte cuvinte schimbarea CP a unui tabel va trebui însoţită de schimbarea cheilor străine care fac referire la aceasta.
- **Simplitatea**. În general, o cheie artificială este mai simplă decât una naturală. Cheile naturale sunt mai complexe, atât dpdv fizic (nr. de octeţi) cât şi al nr. de coloane.
  De exemplu, este mult mai comodă stocarea codului studentului şi utilizarea sa ca cheie străină, decât a combinaţiei dintre numele, prenumele şi data naşterii.

Chei primare. Chei naturale şi chei artificiale

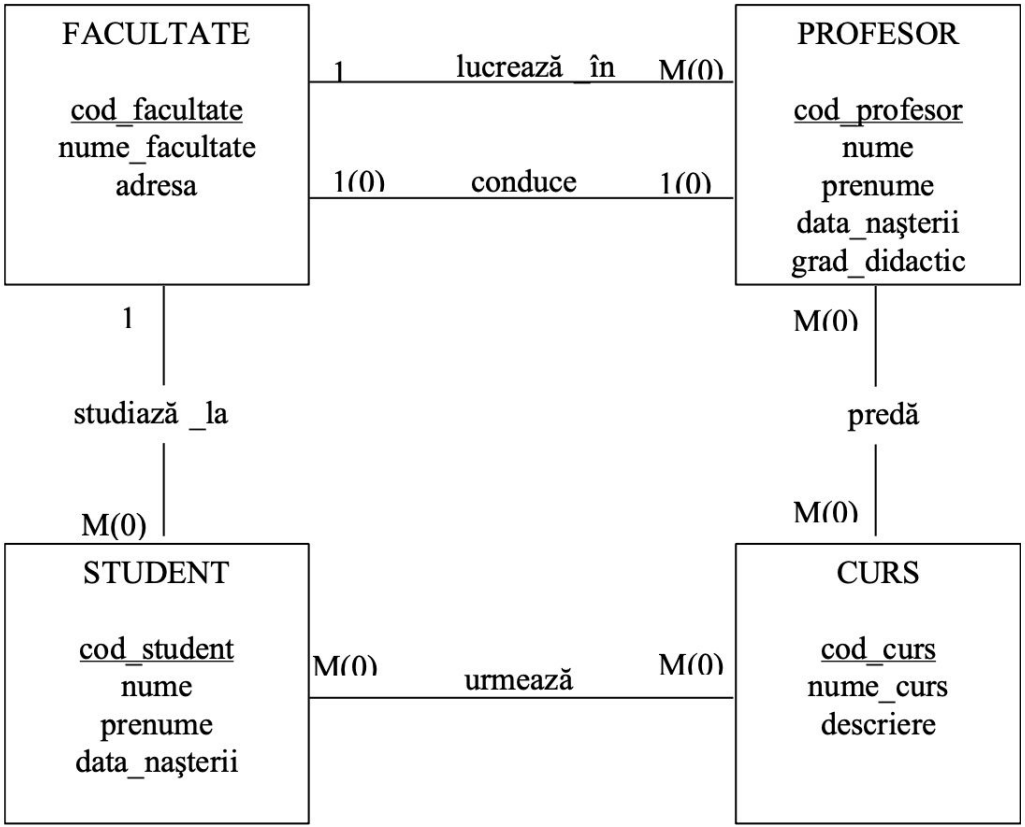
- **Nu prezintă ambiguităţi**. O CP trebuie să nu prezinte ambiguităţi, a. î. să poată fi folosită cu uşurinţă de către dezvoltator sau utilizator în filtrările efectuate pe tabele.
  Şi în această privinţă, o cheie naturală creează probleme.
  De exemplu, numele şi prenumele unui student pot fi formate dintr-unul sau mai multe cuvinte care pot fi despărţite de un spaţiu sau de o linie, etc.

Chei primare. Chei naturale şi chei artificiale

- **Elimină valorile Null**. În cazul CP naturale, valorile Null reprezintă o problemă.
  De exemplu, aceasta înseamnă că un student nu poate fi înregistrat dacă nu i se ştie data de naştere.
- În concluzie, o **CP trebuie să fie unică, diferită de Null, scurtă, simplă, fără ambiguităţi, să nu conţină informaţii descriptive, să fie uşor de manipulat, să fie stabilă şi familiară utilizatorului**.
  Cheile artificiale îndeplinesc toate aceste condiţii în afară de ultima, fiind preferate aproape întotdeauna celor naturale.

Diagrama entitate-legătură (ERD)

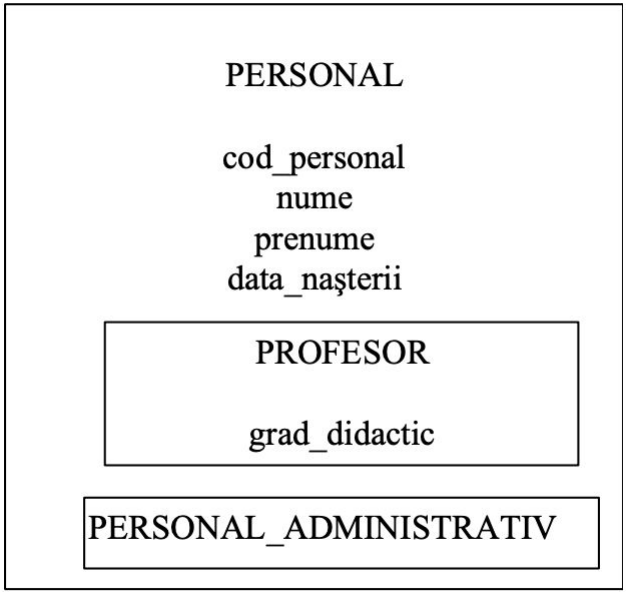
- Entităţile sistemului, împreună cu relaţiile dintre ele se reprezintă prin aşa numita diagramă entitate-legătură (eng. **entity relationship diagram**), în care **entităţile** sunt reprezentate prin **dreptunghiuri**, iar **relaţiile** dintre acestea prin **arce neorientate**, specificându-se şi **cardinalitatea** acestora.
- Pentru fiecare entitate se specifică **CP** şi eventual atributele mai semnificative, atributele care reprezintă CP trebuind să fie **subliniate**.
- Diagrama entitate-legătură a sistemului descris la începutul cursului este reprezentată astfel:


FACULTATE
cod_facultate
nume_facultate
adresa
1
lucrează _în
M(0)
PROFESOR
cod_profesor
nume
prenume
data_naşterii
grad_didactic
1(0)
conduce
1(0)
1
studiază _la
M(0)
M(0)
predă
M(0)
STUDENT
cod_student
nume
prenume
data_naşterii
M(0)
urmează
M(0)
CURS
cod_curs
nume_curs
descriere

Cazuri speciale de entităţi, relaţii şi atribute

- În continuare vom considera câteva cazuri speciale de entităţi, relaţii şi atribute, încercând în acelaşi timp o clasificare a acestora.
- **Subentitate/Superentitate**. O subentitate este o submulţime a unei alte entităţi, numită superentitate. De exemplu, să presupunem că în sistemul prezentat mai înainte nu vom reţine date numai despre profesorii universităţii, ci şi despre tot personalul din universitate. Atunci vom crea o superentitate PERSONAL, pentru care PROFESOR este o subentitate. O altă subentitate a acestei superentităţi va fi PERSONAL_ADMINISTRATIV. O **subentitate** se reprezintă printr-un **dreptunghi inclus în dreptunghiul care reprezintă superentitatea corespunzătoare**, ca în figura următoare:

Subentitate/Superentitate



Cazuri speciale de entităţi, relaţii şi atribute

- CP, atributele şi relaţiile unei superentităţi sunt valabile pentru orice subentitate, reciproca fiind evident falsă.
  De exemplu, CP a entităţii PROFESOR va fi acum „cod_personal”, care este CP a entităţii PERSONAL, în timp ce unele dintre atributele subentităţii PROFESOR (de exemplu „nume”, „prenume”, „data_nasterii”) se regăsesc printre atributele entităţii PERSONAL.
  Pe de altă parte însă, subentitatea PROFESOR poate avea şi alte atribute decât cele specifice superentităţii PERSONAL, de exemplu gradul didactic. Cu alte cuvinte, **atributele comune** vor fi repartizate superentităţii, în timp ce **atributele specifice** vor fi repartizate subetităţilor.

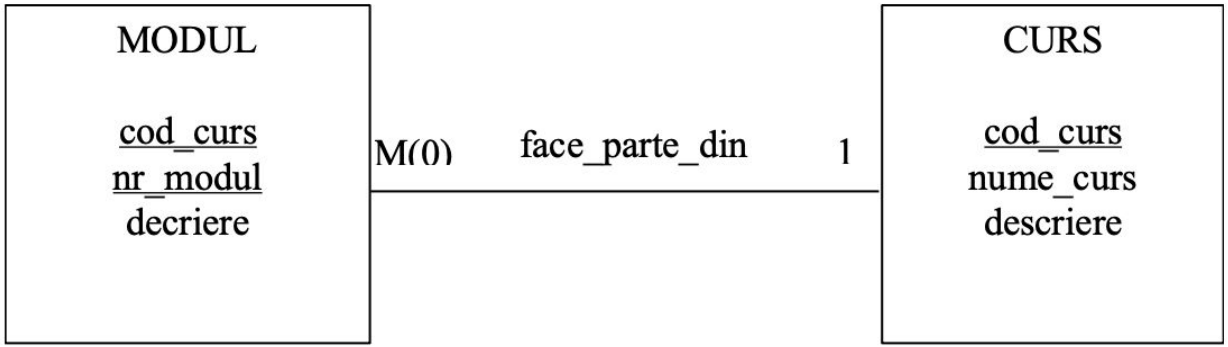
Cazuri speciale de entităţi, relaţii şi atribute

- **Între o subentitate şi superentitatea corespunzătoare există întotdeauna o relaţie 1:1, având cardinalitatea minimă 1:0.**
  Uneori este convenabil să se creeze superentităţi din entităţi cu mai multe atribute comune.
  De exp., din entităţile PROFESOR şi PERSONAL_ADMINISTRATIV s-a creat entitatea PERSOANA. Superentitatea creată va conţine atributele comune, iar cele specifice vor fi repartizate subentităţilor componente.
  În plus, se va crea o nouă cheie artificială pentru superentitatea nou formată. De exemplu, pentru PERSOANA s-a creat un cod personal, care a devenit CP a acestei entităţi.

Cazuri speciale de entităţi, relaţii şi atribute

- **Entitate dependentă (detaliu)/entitate master**.
  O **entitate dependentă** (detaliu) este o entitate care **nu poate exista de sine stătătoare, ci numai ataşată unei alte entităţi**, aceasta din urmă fiind numită entitatea master a acestei legături.
  De exemplu, dacă presupunem că fiecare curs poate fi constituit dintr-unul sau mai multe module, atunci entitatea MODUL va fi o entitate dependentă de CURS, ca în figura:

Entitate dependentă (detaliu)/entitate master

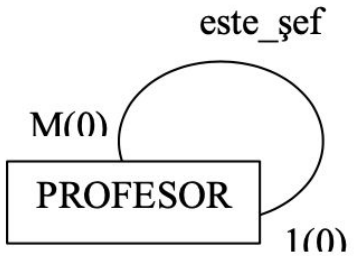


Cazuri speciale de entităţi, relaţii şi atribute

- **Între entităţile master şi detaliu va exista întotdeauna o relaţie 1:N, având cardinalitatea minimă 1:0**.
  CP a unei entităţi detaliu va fi formată din CP a entităţii master plus una sau mai multe atribute ale entităţii detaliu.
  De exemplu, cheia entităţii MODUL poate fi aleasă ca fiind combinaţia dintre cod_curs şi nr_modul, acesta din urmă specificând numărul de ordine al unui modul în cadrul unui curs.

Cazuri speciale de entităţi, relaţii şi atribute

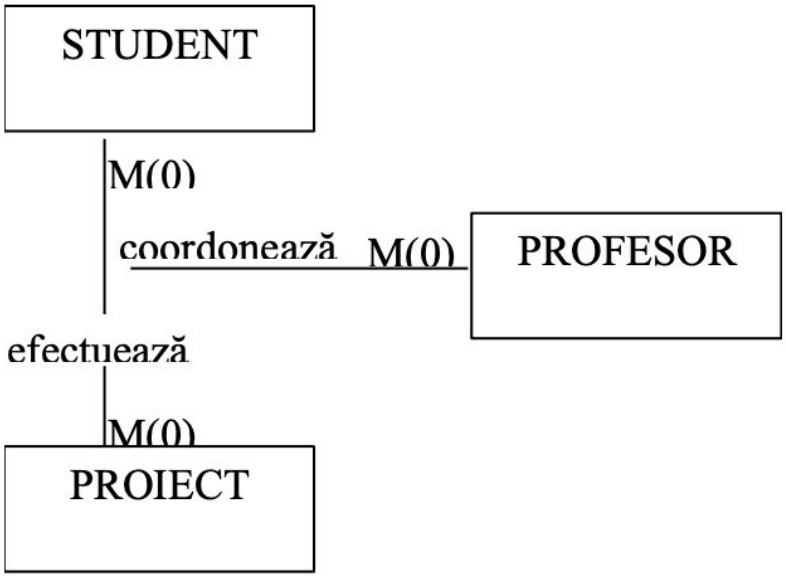
- **Relaţii recursive.**
  Pot exista **relaţii** nu numai între două entităţi diferite, ci şi **între o entitate şi ea însăşi**; acestea se numesc relaţii recursive.
  De exemplu, dacă presupunem că activitatea de cercetare în universitate este organizată pe o structură ierarhică, adică un profesor poate avea un şef şi poate fi la rândul lui şeful mai multor profesori, atunci entitatea profesor admite o relaţie recursivă de tipul N:1, ca în figura



Cazuri speciale de entităţi, relaţii şi atribute

- **Relaţii binare (de tip 2)/ relaţii între mai mult de două entităţi (de tip 3)**.
  Până acum am discutat doar despre relaţiile dintre 2 entităţi, numite relaţii binare sau de tip 2.
  Pot însă exista şi relaţii între mai mult de două entităţi, pe care le vom numi relaţii de tip 3.
  De exemplu, să presupunem că fiecare student trebuie să efectueze mai multe proiecte, iar pentru fiecare proiect el poate să-şi aleagă unul sau mai mulți profesori coordonatori. Deci relaţia „efectuează_coordonează” este o relaţie de tip 3 între entităţile STUDENT, PROIECT şi PROFESOR, ca în figura:

Relaţii binare (de tip 2)/ relaţii între mai mult de două entităţi (de tip 3)



Cazuri speciale de entităţi, relaţii şi atribute

- **O relaţie de tip 3 nu poate fi spartă în relaţii binare între entităţile componente**, un exemplu este oferit în figura următoare, unde prin spargerea relaţiei „efectuează_coordonează” în 3 relaţii binare prin proiecţie se obţin informaţii eronate, relaţia iniţială nemaiputând fi reconstituită din relaţiile componente.

STUDENT	PROIECT	PROFESOR
s1	p1	x2
s1	p2	x1
s2	p1	x1

a) Relaţia de tip 3 iniţială

STUDENT	PROIECT
s1	p1
s1	p2
s2	p1

STUDENT	PROFESOR
s1	x2
s1	x1
s2	x1

PROIECT	PROFESOR
p1	x2
p2	x1
p1	x1

b) Descompunerea relaţiei de tip 3 în 3 relaţii binare prin proiecţie

STUDENT	PROIECT	PROFESOR
s1	p1	x1
s1	p1	x2
s1	p2	x1
s2	p1	x1

c) Reconstituirea eronată a relaţiei iniţiale

Cazuri speciale de entităţi, relaţii şi atribute

- **Atribute simple/ compuse/ repetitive (multivaloare)/ calculate (deduse)**.
  Atributele pot fi de 4 feluri:
  - simple,
  - compuse,
  - repetitive (multivaloare) şi
  - calculate (deduse).
- Unui atribut simplu îi corespunde o singură valoare, atomică.
  De exemplu, numele şi prenumele unui student sunt atribute simple.

Cazuri speciale de entităţi, relaţii şi atribute

- Un atribut compus este format din mai multe atribute simple, numite componentele sale.
- Valoarea unui atribut compus este reprezentată de valorile atributelor componente.
  Dacă presupunem, de exemplu, că o adresă se poate descompune în componentele ţară, oraş, stradă, număr şi cod, atunci adresa este un atribut compus din 5 componente.

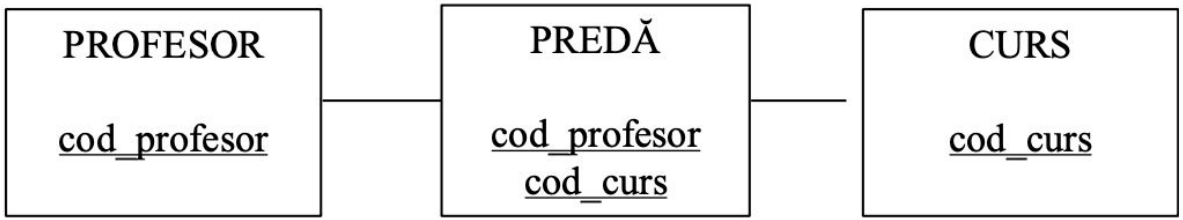
Cazuri speciale de entităţi, relaţii şi atribute

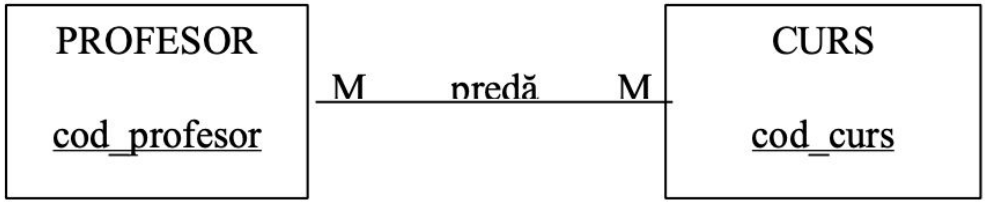
- Un **atribut repetitiv (multivaloare)** este un atribut care poate avea mai multe valori, numărul acestora variind de la o instanţă la alta.
  De exemplu, un student poate avea mai multe numere de telefon, deci acesta este un atribut repetitiv.
- Un **atribut calculat** reprezintă un atribut a cărui valoare nu este cunoscută direct, ci calculată pe baza valorilor altor atribute.
  De exemplu atributul valoare este calculat ca produs între atributele cantitate şi preţ. **Atributele calculate se folosesc foarte rar deoarece ele reprezintă de fapt o redundanţă a datelor**.

Probleme în identificarea entităţilor, relaţiilor şi atributelor

- **Relaţie sau entitate?**
  Uneori este greu de identificat dacă o componentă a sistemului este relaţie sau entitate.
  Dacă o entitate are o cheie provenită din combinaţia cheilor primare a două sau mai multe entităţi, atunci trebuie definită o relaţie.
  Deci entitatea PREDĂ va avea semnificaţia unei relaţii între entităţile PROFESOR şi CURS, reprezentată astfel:

Relaţie sau entitate?

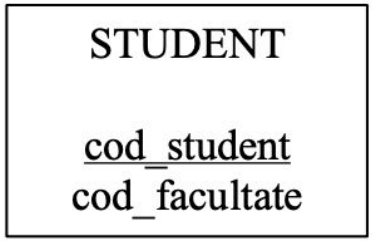


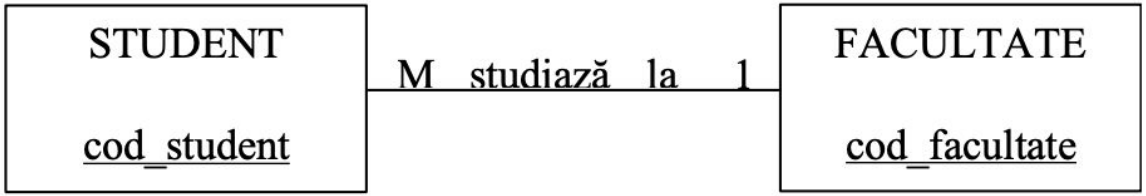


Probleme în identificarea entităţilor, relaţiilor şi atributelor

- **Relaţie sau atribut?**
- După cum am văzut până acum, o relaţie poate fi reprezentată ca un atribut al unei entităţi, după cum atributele unei entităţi pot fi înlocuite cu relaţii.
  Deci, care este diferenţa între o relaţie şi un atribut?
  Atunci când un atribut al unei entităţi reprezintă CP a altei entităţi, el face referinţă la o relaţie. Deci atributul cod_facultate din prima entitate STUDENT va reprezenta o relaţie între entităţile STUDENT şi FACULTATE, așa cum se poate vedea în figura:

Relaţie sau atribut?



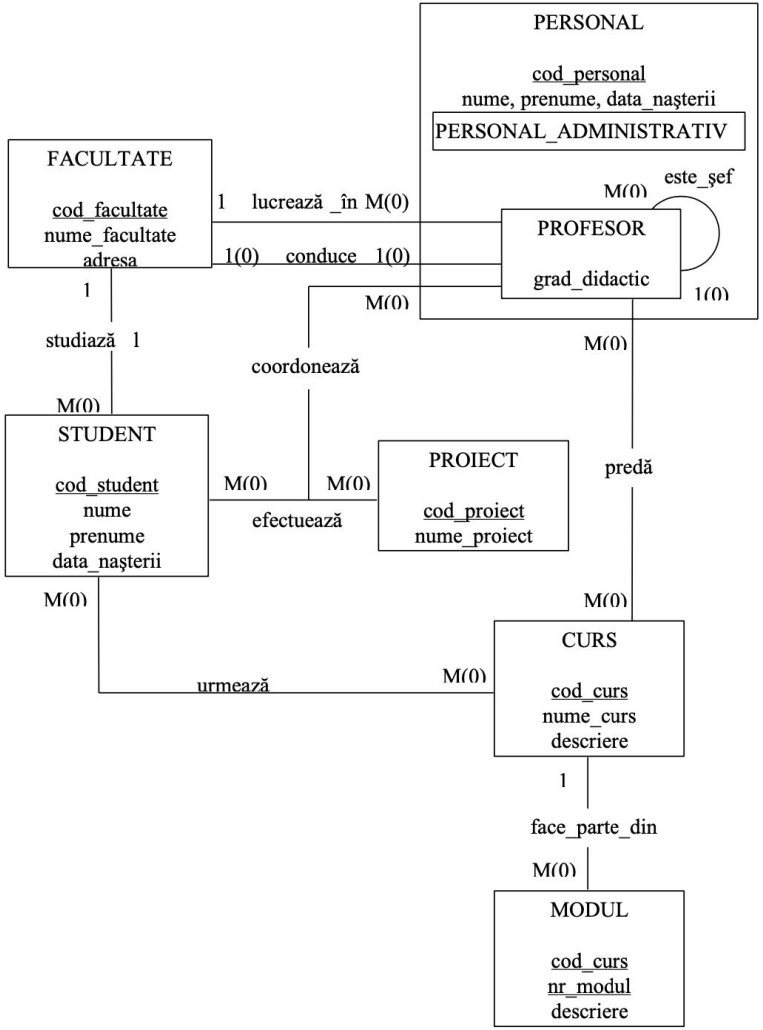


Etapele obţinerii modelului entitate-legătură

- Pentru realizarea modelului entitate-legătură al sistemului analizat sunt parcurse următoarele etape:
  - Identificarea **entităţilor** sistemului.,
  - Identificarea **relaţiilor** sistemului şi stabilirea **cardinalităţii** acestora.
  - Identificarea **atributelor** entităţilor şi relaţiilor sistemului.
  - Stabilirea **cheilor primare** ale entităţilor.
  - Trasarea **diagramei entitate-legătură**.

Exemplu de ERD

Diagrama entitate-legătură a sistemului prezentat ca exemplu în acest curs, incluzând entităţile şi relaţiile menţionate mai înainte:



Etapele obţinerii modelului entitate-legătură

- Trebuie remarcat că aceeaşi realitate poate fi percepută diferit de către analişti diferiţi, aşa că **este posibilă obţinerea de modele diferite pentru acelaşi sistem**, după cum şi un sistem poate să se modifice în timp, ceea ce va atrage la rândul său modificarea modelului asociat.
- În sfârşit, există şi alte moduri grafice de reprezentare a diagramei entitate-legătură, cum ar fi aceea din fișierul diagrameER.pdf, în acest curs prezentându-se doar una dintre notaţiile existente.

Bibliografie

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