

BAZE DE DATE

CURS 1

NOTARE

- ▶ **Modalitatea de evaluare și cerințele**
 - ▶ **Fișiere disponibile pe Teams / Moodle după primul curs**

OPORTUNITĂȚI

ORACLE® Academy

► Conturi individuale pe platforma

<http://ilearning.oracle.com>

- Cursuri de Java și baze de date
- Quizz-uri
- Certificări junior

→ Cerere prin email: letitia.marin@unibuc.ro

PLAN CURS

- 1. Generalități despre baze de date (structură, evoluție, caracteristici, funcționalități, perspective etc.).**
- 2. Proiectarea (*design-ul*) bazelor de date:**
 - ▶ proiectare diagrame *E/R*
 - ▶ modelul relațional, diagrame conceptuale, *UML*
 - ▶ prelucrarea și optimizarea cererilor, normalizare, denormalizare, regulile lui Codd etc.
- 3. Introducere în bazele de date nerelaționale.**
- 4. Limbaje pentru gestionarea datelor. Neprocedural în baze de date (standardul *SQL*).**

Bibliografie

- ▶ Connolly, T.M., Begg, C.E., Database Systems: A Practical Approach to Design, Implementation and Management, 6th edition, Pearson Education, 2014
- ▶ Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., Programare avansată în Oracle9i, Editura Tehnică, Bucureşti, 2004.
- ▶ Popescu, I., Velcescu, L., ***Proiectarea bazelor de date***, Editura Universităţii din Bucureşti, 2008
- ▶ Popescu, I., Velcescu, L., ***Naprocedural în Oracle 10g***, Editura Universităţii din Bucureşti, 2008

INTRODUCERE

- ▶ **Bază de date**
- ▶ **Sistem de gestiune a bazelor de date**
- ▶ **Dicționarul datelor**

INTRODUCERE

BD

- ▶ **Ce este o bază de date (BD) ?**
- ▶ **Exemple?**
- ▶ **Cui aparțin datele?**
- ▶ **Cine le poate accesa?**

INTRODUCERE

BD

- ▶ **BAZA DE DATE** = ansamblu **structurat** de date **coerente, fără redundanță inutilă**, astfel încât acestea pot fi prelucrate eficient de mai mulți utilizatori într-un **mod concurrent**
- ▶ Colecție de date **persistente**, care sunt folosite de către sistemele de aplicații ale unei anumite „întreprinderi”

INTRODUCERE

BD

- ▶ „Întreprindere“ (eng. *Enterprise*):
 - **Reguli** proprii de funcționare
 - **Mulțime de date** referitoare la modul său de operare
- ▶ **Datele din BD:**
 - **Integrate**
 - **Partajate**

INTRODUCERE

SGBD

- ▶ **Ce este un Sistem de Gestiune a Bazelor de Date?**
- ▶ **Exemple?**

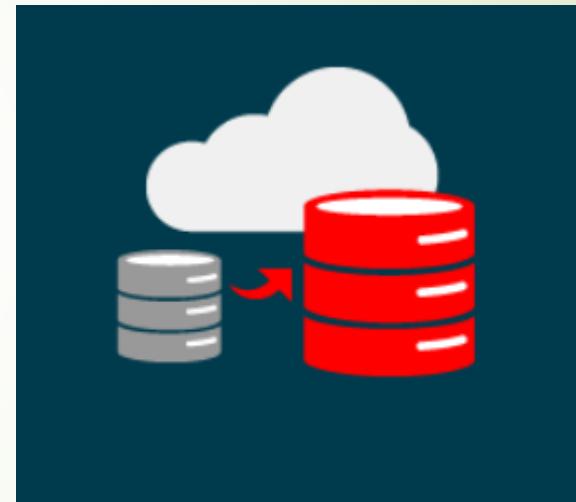
INTRODUCERE

SGBD

- ▶ **Sistem de Gestiune a Bazelor de Date (SGBD = DBMS – Data Base Management System)** este **un produs software** care asigură interacțiunea cu o bază de date
 - ▶ definirea
 - ▶ consultarea
 - ▶ actualizarea datelor din baza de date.
- ▶ Cererile de **acces la baza de date** sunt tratate și controlate de către **SGBD**.

INTRODUCERE

BD



<https://www.oracle.com/database/technologies/upgrades.html>

INTRODUCERE

Dicționarul datelor

- ▶ **Dicționarul datelor** (catalog de sistem) = metabază de date
 - ▶ „date despre date“
 - ▶ structurat și administrat ca o bază de date
- ▶ **Ce informații regăsim în DD?**
- ▶ **Cum obținem aceste informații?**

INTRODUCERE

Dicționarul datelor

- ▶ DD conține:
 - ▶ descrierea tuturor obiectelor unei baze de date
 - ▶ starea acestor obiecte
 - ▶ diversele constrângeri de securitate și de integritate etc.
- ▶ DD poate fi interogat ca orice altă bază de date.

INTRODUCERE

DBA

- ▶ **Administratorul bazei de date** (DBA – Data Base Administrator) = persoană sau un grup de persoane ce răspund de ansamblul activităților legate de baza de date.
 - ▶ analiză
 - ▶ proiectare
 - ▶ implementare
 - ▶ exploatare
 - ▶ întreținere etc.

INTRODUCERE

DBA

- ▶ Categorii de atribuții DBA:
 - ▶ atribuții de **proiectare**,
 - ▶ atribuții **administrative**,
 - ▶ atribuții **operative** și
 - ▶ atribuții de **coordonare**.

GESTIUNEA BAZELOR DE DATE

- ▶ Arhitectura unui **sistem de baze de date**:
 1. **baza de date propriu-zisă** în care se memorează datele;
 2. **sistemul de gestiune a bazei de date**, care realizează gestionarea și prelucrarea complexă a datelor;
 3. **un dicționar al bazei de date** (metabaza de date), ce conține informații despre date, structura acestora, statistici, documentație;
 4. **mijloace hardware** (comune sau specializate);
 5. **reglementări administrative** destinate bunei funcționări a sistemului;
 6. **personalul implicat** (utilizatori finali, administratorul datelor, administratorul bazei de date, proiectanți / programatori de aplicații) → **4 categorii de utilizatori**

GESTIUNEA BAZELOR DE DATE

- ▶ **Administratorul de date** (DA) este un manager
 - ▶ decide care date trebuie stocate în baza de date;
 - ▶ stabilește regulile de întreținere și de tratare a acestor date după ce sunt stocate.

GESTIUNEA BAZELOR DE DATE

- ▶ **Administratorul bazei de date** (DBA) este un profesionist în domeniul IT, care:
 - ▶ creează baza de date reală;
 - ▶ implementează elementele tehnice de control;
 - ▶ este responsabil cu asigurarea funcționării sistemului la performanțe adecvate, cu monitorizarea performanțelor;
 - ▶ furnizează diverse servicii tehnice etc.

GESTIUNEA BAZELOR DE DATE

- ▶ **Proiectanții de baze de date** pot acoperi 2 aspecte ale concepției (design-ului):
 - ▶ Proiectare fizică
 - ▶ Proiectare logică

GESTIUNEA BAZELOR DE DATE

- ▶ **Proiectarea logică** presupune o cunoaștere completă și amănunțită:
 - ▶ a **modelului real** de proiectat și
 - ▶ a **regulilor de funcționare** a acestuia.
- ▶ Proiectarea **conceptuală** a bazei de date
- ▶ Modelul creat este **independent** de programele de aplicații, de limbajele de programare
- ▶ Proiectarea logică a bazei de date, îndreptată spre un anumit **model de date** (relațional, orientat obiect, ierarhic etc.).

GESTIUNEA BAZELOR DE DATE

► **Proiectarea fizică:**

- ▶ preia modelul logic de date
- ▶ stabilește cum va fi realizat fizic
- ▶ presupune cunoașterea funcționalităților SGBD-ului, avantajele și dezavantajele fiecărei alternative.
- ▶ Transpunerea modelului logic într-un set de **tabele** supuse unor **constrângeri**, se selectează **structuri de stocare** și **metode de acces** specifice, astfel încât să se asigure **performanțe**, se iau măsuri privind **securitatea** datelor.

GESTIUNEA BAZELOR DE DATE

► Cerințe minime BD:

- ▶ redundanță minimă în date;
- ▶ furnizarea în **timp util** a informațiilor solicitate (timpul de răspuns la o interogare);
- ▶ asigurarea unor **costuri minime** în prelucrarea și întreținerea informației;
- ▶ capacitatea de a satisface, cu aceleași 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**);

GESTIUNEA BAZELOR DE DATE

► Cerințe minime BD (continuare):

- ▶ asigurarea **securității datelor** prin mecanisme de protecție împotriva accesului neautorizat (confidențialitate);
- ▶ înglobarea unor facilități destinate validării datelor și 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 și anticiparea nevoilor viitoare (**compatibilitate și expandabilitate**);
- ▶ permisivitatea, prin ierarhizarea datelor după criteriul frecvenței acceselor, a unor reorganizări (eventual dinamice) care sporesc performanțele bazei.

GESTIUNEA BAZELOR DE DATE

- ▶ **Patru niveluri de abstractizare și de percepție a datelor:**
 - ▶ intern (fizic)
 - ▶ conceptual
 - ▶ logic
 - ▶ extern.
- ▶ Datele există doar la nivel **fizic**, iar celelalte trei niveluri reprezintă **virtualizări** ale acestora.

GESTIUNEA BAZELOR DE DATE

- ▶ **Nivelul fizic (intern)** → schema fizică a datelor (bit, octet, adresă);
- ▶ **Nivelul conceptual** → schema conceptuală a datelor (articol, înregistrare, zonă) și reprezintă **viziunea programatorilor de sistem** asupra datelor;
- ▶ **Nivelul logic** → una din schemele logice posibile ale datelor și reprezintă **viziunea programatorului de aplicație** asupra datelor;
- ▶ **Nivelul virtual (extern)** reprezintă **viziunea utilizatorului final** asupra datelor.

GESTIUNEA BAZELOR DE DATE

- ▶ **Independența datelor** → două aspecte fundamentale:
 - ▶ o modificare a structurii fizice nu va afecta aplicația și
 - ▶ reciproc, modificări ale aplicației vor lăsa nealterată structura fizică de date.
- ▶ **Independența fizică**
- ▶ **Independența logică**

GESTIUNEA BAZELOR DE DATE

- ▶ **Independență fizică:** posibilitatea **modificării schemei fizice** a datelor fără ca aceasta să implice modificarea schemei conceptuale, a schemei logice și a programelor de aplicație.
- ▶ **Independență logică:** posibilitatea **modificării schemei conceptuale** a datelor fără ca aceasta să implice modificarea schemei logice și a programelor de aplicație.
 - ▶ Independența logică \leftrightarrow fiecare utilizator are iluzia că este singurul beneficiar al unor date pe care, în realitate, le folosește în comun cu alți utilizatori.

GESTIUNEA BAZELOR DE DATE

- ▶ **Independență 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.

GESTIUNEA BAZELOR DE DATE

- ▶ **Limbaje pentru baze de date**
 - ▶ **Limbaje pentru definirea datelor** (LDD – *Data Description Language*)
 - ▶ **Limbaje pentru prelucrarea datelor** (LMD – *Data Manipulation Language*)
 - ▶ **Limbaje pentru controlul datelor** (LCD – *Data Control Language*)

GESTIUNEA BAZELOR DE DATE

- ▶ Limbaje pentru definirea datelor (LDD – *Data Description Language*).
 - ▶ definirea **entităților** și a **atributelor** acestora
 - ▶ sunt precizate **relațiile** dintre date și strategiile de acces la ele
 - ▶ sunt stabilite criterii diferențiate de **confidențialitate** și de validare automată a datelor utilizate.

GESTIUNEA BAZELOR DE DATE

- ▶ Limbaje pentru prelucrarea datelor (LMD – *Data Manipulation Language*).
 - ▶ o comandă are următoarea structură: operația, criterii de selecție, mod de acces (secvențial, indexat etc.).
 - ▶ există limbaje LMD:
 - ▶ **procedurale**, care specifică **cum** se obține rezultatul unei comenzi LMD și
 - ▶ **neprocedurale**, care descriu doar datele **ce** vor fi obținute și nu modalitatea de obținere a acestora.

GESTIUNEA BAZELOR DE DATE

- ▶ Limbaje pentru controlul datelor (LCD – *Data Control Language*).
 - ▶ asigurarea **confidențialității** și **integrității** datelor
 - ▶ salvarea informației în cazul unor **defecțiuni**
 - ▶ rezolvarea unor probleme de **concurență**.

GESTIUNEA BAZELOR DE DATE

► Obiectivele unui SGBD

- Independența fizică
 - independența structurilor de stocare în raport cu structurile de date din lumea reală
- Independența logică
 - fiecare grup de lucru poate să cunoască doar o parte a semanticii datelor, să vadă doar o submulțime a datelor și numai sub forma în care le dorește
- Prelucrarea datelor de către neinformaticieni
- Administrarea centralizată a datelor
- Coerența datelor
 - Informația trebuie să satisfacă constrângeri statice sau dinamice, locale sau generale.

GESTIUNEA BAZELOR DE DATE

► Obiectivele unui SGBD (continuare)

► Neredundanța datelor

- Administrarea coerentă a datelor trebuie să asigure neduplicarea fizică a datelor.
- Pentru a realiza performanțe referitoare la timpul de acces la date și răspuns la solicitările utilizatorilor, se acceptă o anumită redundanță a datelor.

► Partajabilitatea datelor.

- Aplicațiile pot să partajeze datele din baza de date în timp și simultan.
- O aplicație poate folosi date ca și cum ar fi singura care le utilizează, fără a ști că altă aplicație, concurrent, le poate modifica

GESTIUNEA BAZELOR DE DATE

- ▶ **Obiectivele unui SGBD** (continuare)
 - ▶ **Securitatea și confidențialitatea datelor.**
 - ▶ datele trebuie protejate de un acces neautorizat sau rău intenționat
 - ▶ există mecanisme care permit identificarea și **autentificarea** utilizatorilor
 - ▶ există proceduri de **acces autorizat** care depind de date și de utilizator.
 - ▶ Sistemul de gestiune trebuie să asigure securitatea fizică și logică a informației și să garanteze că numai utilizatorii autorizați pot efectua operații corecte asupra bazei de date.

GESTIUNEA BAZELOR DE DATE

► Dezavantajele SGBD-urilor

- **complexitatea și dimensiunea sistemelor** pot să crească considerabil, datorită necesității extinderii funcționalităților sistemului;
- **costul**, care variază în funcție de mediu și funcționalitatea oferită, la care se adaugă 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* și *software*;
- **impactul unei defecțiuni** asupra aplicațiilor, bazei de date sau sistemului de gestiune.

GESTIUNEA BAZELOR DE DATE

► Structura unui SGBD

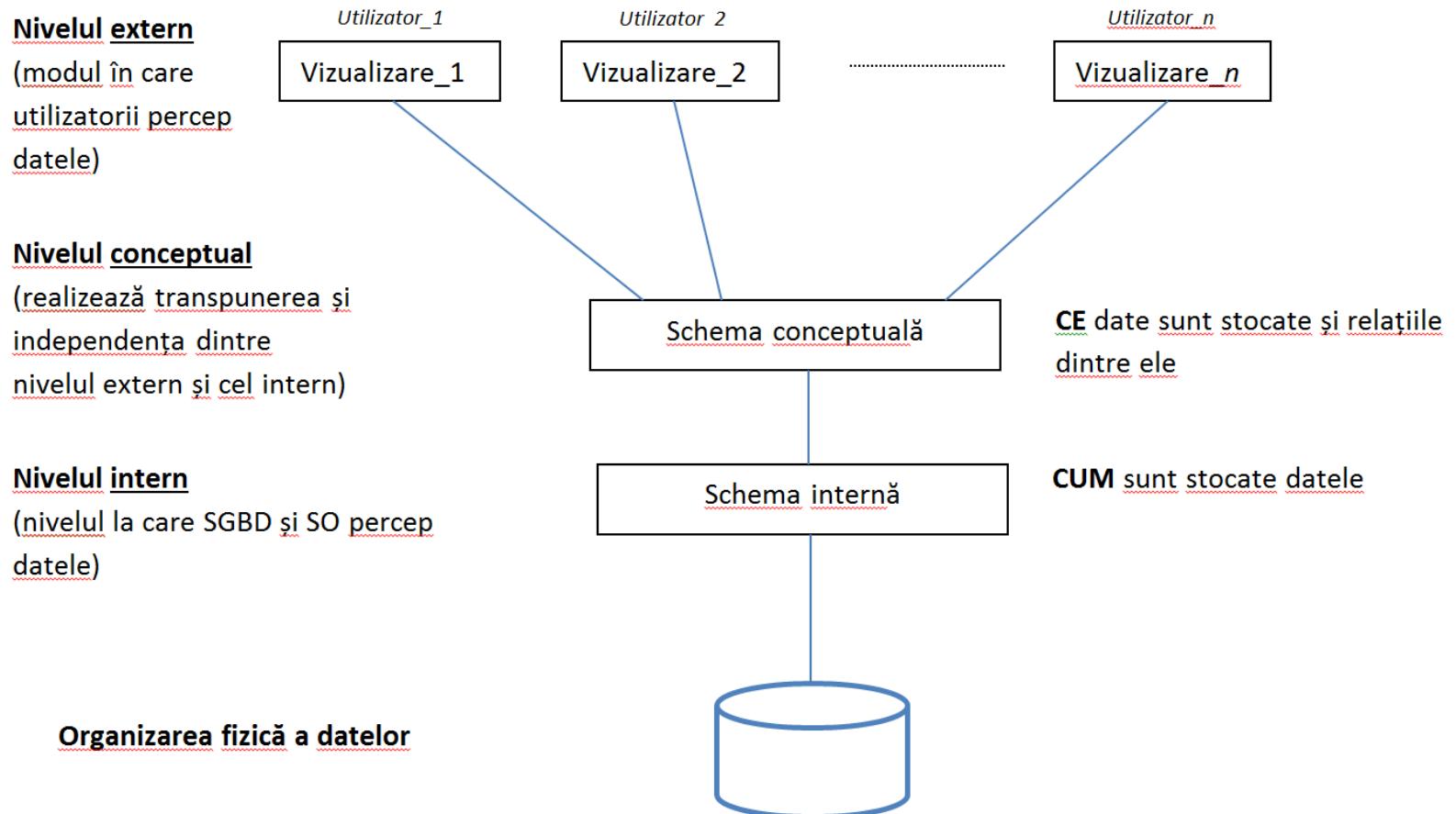
- ▶ complexitate variabilă
- ▶ nivelul real de funcționalitate diferă de la produs la produs
- ▶ cel puțin 5 clase de module

GESTIUNEA BAZELOR DE DATE

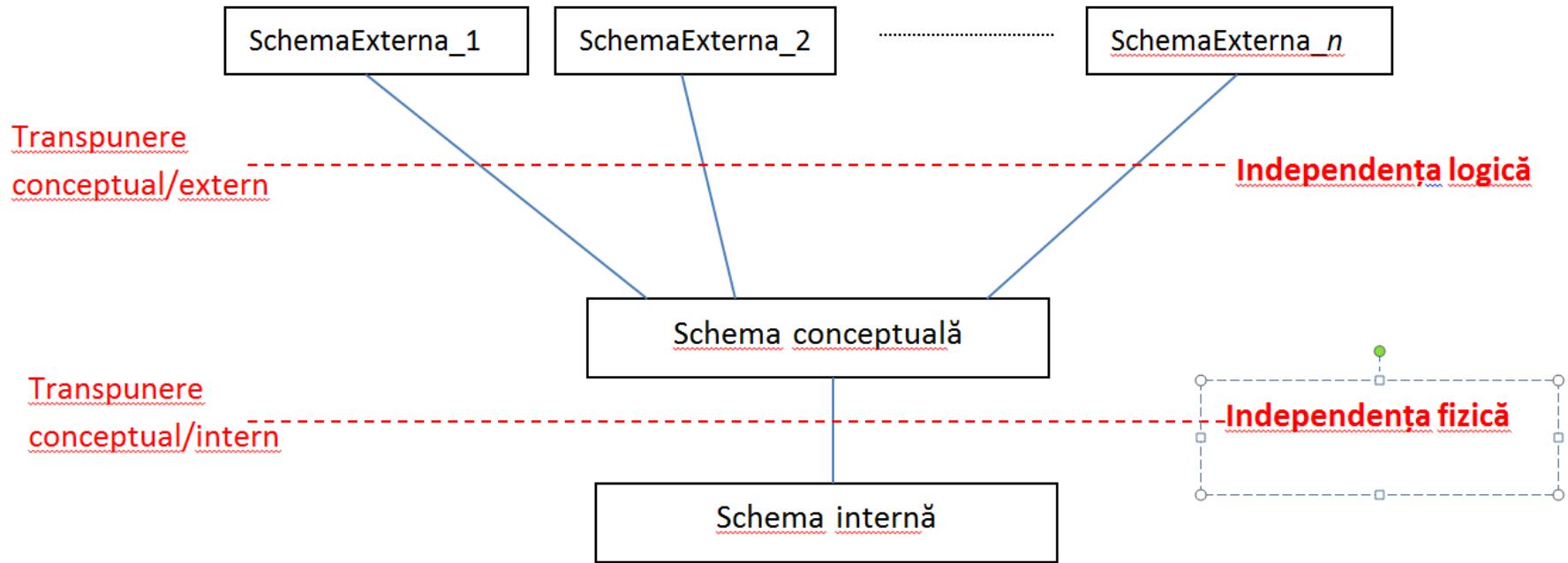
► **Module SGBD:**

- ▶ **programe de gestiune a bazei de date (PGBD)**, care realizează accesul fizic la date ca urmare a unei comenzi;
- ▶ **module pentru tratarea LDD**, ce permit traducerea unor informații în obiecte ce pot fi apoi exploataate în manieră procedurală sau neprocedurală;
- ▶ **module pentru tratarea LMD** care permit utilizatorilor inserarea, ștergerea, reactualizarea sau consultarea informației dintr-o bază de date;
- ▶ **module utilitare**, care asigură întreținerea, prelucrarea, exploatarea corectă și ușoară a bazei de date;
- ▶ **module de control**, care permit controlul programelor de aplicație, asigurarea confidențialității și integrității datelor, rezolvarea unor probleme de concurență, recuperarea informației în cazul unor avarii sau defecțiuni *hardware* sau *software* etc.

Independență logică și fizică



Independență logică și fizică



Independență logică \Leftrightarrow imunitatea schemelor externe față de modificările efectuate în schema conceptuală

Independență fizică \Leftrightarrow imunitatea schemei conceptuale față de modificările efectuate în schema internă

Corespondența extern-extern permite definirea unei vizualizări în funcție de altele, fără a necesita o definiție explicită a corespondenței cu nivelul conceptual.

Arhitectura sistemelor de gestiune a bazelor de date

- ▶ **Independență fizică și logică a datelor** => 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 sistemelor de gestiune a bazelor de date

- ▶ **Nivelul central -> nivelul conceptual**
 - ▶ structura canonică a datelor ce caracterizează procesul de modelat (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 și legăturile între aceste entități
 - ▶ definirea **regulilor** pe care trebuie să le respecte datele etc.

Arhitectura sistemelor de gestiune a bazelor de date

- ▶ **Nivelul intern** = structura internă de stocare a datelor
- ▶ **Schema internă** permite:
 - ▶ descrierea datelor unei baze sub forma în care sunt stocate în memoria calculatorului
- ▶ *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.*

Arhitectura sistemelor de gestiune a bazelor de date

► Nivel extern

- 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 sistemelor de gestiune a bazelor de date

- ▶ Corespondențe între niveluri
 - ▶ **corespondența conceptual-intern** → modul în care înregistrările și câmpurile conceptuale sunt reprezentate la nivel intern;
 - ▶ **corespondența extern-conceptual** → cheia independenței logice de 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 sistemelor de gestiune a bazelor de date

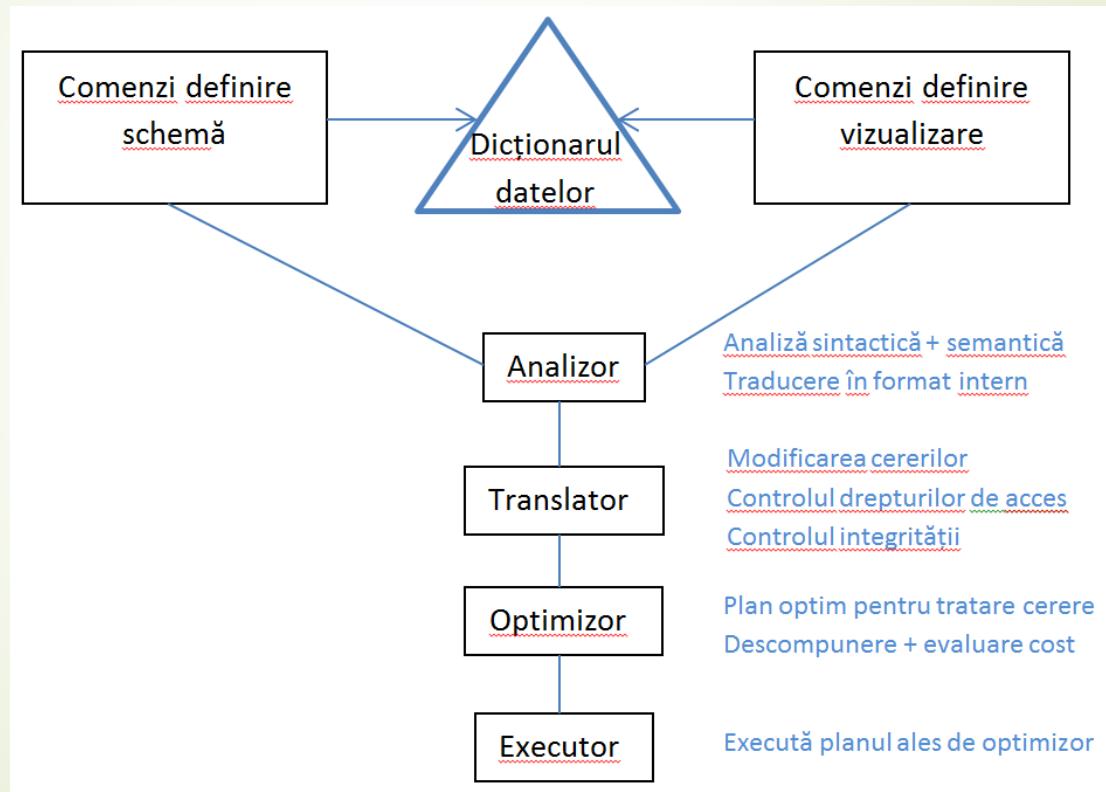
- ▶ **Arhitectura funcțională de referință** propusă de grupul de lucru **ANSI/X3/SPARC** este axată pe **dicționarul datelor** și cuprinde două părți:
 - ▶ prima, permite **descrierea datelor** (compoziția dicționarului datelor);
 - ▶ a doua, permite **prelucrarea datelor** (interrogarea și reactualizarea bazei de date).

Figura – curs Moodle!

Arhitectura sistemelor de gestiune a bazelor de date

- ▶ **Gardarin** a propus o arhitectură funcțională apropiată de arhitectura sistemelor de gestiune actuale.
- ▶ Arhitectura are la bază doar două niveluri:
 - ▶ **schema**, care corespunde integrării schemelor interne și conceptuale;
 - ▶ **vizualizarea**, care este o schemă externă.

Arhitectura sistemelor de gestiune a bazelor de date



EVOLUȚIA BAZELOR DE DATE

- Istoria BD și SGBD poate fi rezumată în **trei generații**:
 - sisteme **ierarhice și rețea**,
 - sisteme **relaționale**,
 - sisteme **avansate** (orientate obiect, **relaționale orientate obiect**, deductive, multimedia, multibaze, active, temporale, **distribuite, decizionale, magazii de date** etc.).

EVOLUȚIA BAZELOR DE DATE

Baze de date ierarhice și rețea

- ▶ datele sunt reprezentate la nivel de articol prin legături ierarhice (arbore) sau de tip graf
- ▶ slaba independentă fizică a datelor complică administrarea și prelucrarea acestora
- ▶ limbajul de prelucrare a datelor impune programatorului să specifice drumurile de acces la date.

EVOLUȚIA BAZELOR DE DATE

Baze de date relaționale

- ▶ Modelul relațional (1970) tratează entitățile ca niște relații. Piața actuală de baze de date este acoperită în majoritate de sisteme **relaționale**.
- ▶ Bazele de date relaționale sunt caracterizate de:
 - ▶ structuri de date simple, intuitive,
 - ▶ inexistența pointerilor vizibili pentru utilizator,
 - ▶ constrângerile de integritate,
 - ▶ operatori aplicați relațiilor care permit definirea, căutarea și reactualizarea datelor.

EVOLUȚIA BAZELOR DE DATE

- ▶ Bazele de date relaționale nu folosesc obiecte complexe și dinamice
- ▶ Nu realizează gestiunea datelor distribuite
- ▶ Nu realizează gestiunea cunoștințelor
- ▶ A treia generație de SGBD-uri, **sistemele avansate**, încearcă să depășească aceste limite ale sistemului relațional.

EVOLUȚIA BAZELOR DE DATE

Baze de date orientate obiect

- ▶ Ce nu realizează BD relaționale?
 - ▶ Suportul obiectelor complexe și dinamice, prelucrarea acestora
 - ▶ 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 și la refacerea acestuia prin compunere.
 - ▶ Limbajele modelului relațional permit prelucrarea cu dificultate a obiectelor complexe.
 - ▶ Un sistem relațional nu suportă obiecte dinamice care încorporează atât partea de date (informații) efective, cât și o parte relativă la tratarea acestora.
- ▶ Îmbinarea tehnicii limbajelor orientate obiect cu a bazelor de date a permis realizarea **bazelor de date orientate obiect**.

EVOLUȚIA BAZELOR DE DATE

Baze de date orientate obiect (continuare)

- ▶ Avantaje SGBDOO:
 - ▶ realizează o **modelare superioară a informației**,
 - ▶ furnizează posibilități superioare de **deducție** (ierarhie de clase, moștenire),
 - ▶ permit luarea în considerare a **aspectelor dinamice** și integrarea descrierii structurale și comportamentale.
- ▶ Dezavantaje:
 - ▶ absența unui SGBDOO de referință;
 - ▶ gestiunea obiectelor complexe este mai dificilă;
 - ▶ utilizatorii au investit sume uriașe în sistemele relaționale și nu le pot abandona cu ușurință. Trecerea la tehnologia orientată obiect implică investiții mari și nu păstrează aproape nimic din vechile soluții.

EVOLUȚIA BAZELOR DE DATE

Baze de date relaționale orientate obiect

- ▶ Simplitatea modelului relațional + puterea tehnologiei orientate obiect => baze de date relaționale orientate obiect.
- ▶ Construcția unui sistem de gestiune de baze de date relaționale orientate obiect (SGBDROO) trebuie să pornească de la cele 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 și adăugând caracteristicile relaționale.

EVOLUȚIA BAZELOR DE DATE

► Baze de date deductive

- ▶ O relație este o mulțime de înregistrări ce reprezintă **fapte**.
- ▶ Cunoștințele sunt **aserțiuni** generale și abstracte asupra faptelor.
- ▶ Cunoștințele permit să raționezi, ceea ce permite **deducerea** de noi fapte, plecând de la fapte cunoscute.
- ▶ Un SGBD relațional suportă o formă limitată de cunoștințe, și anume constrângerile de integritate, iar restul trebuie integrate în programele de aplicație.

EVOLUȚIA BAZELOR DE DATE

- ▶ **Baze de date deductive** (continuare)
 - ▶ Prin programarea logică se **gestionează cunoștințe relativ la baze de date** care, în general, sunt **relaționale**.
 - ▶ Deducerea de noi informații, plecând de la informațiile stocate în baza de date.

EVOLUȚIA BAZELOR DE DATE

► **Baze de date deductive** (continuare)

► Un SGBD deductiv posedă:

- un **limbaj de definire a datelor** care permite definirea structurii predicatelor sub formă de relații și constrângeri de integritate asociate;
- un **limbaj de prelucrare a datelor** care permite efectuarea reactualizărilor asupra datelor și 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

Baze de date distribuite

- ▶ Sistem distribuit = ansamblu de mașini ce sunt interconectate printr-o rețea de comunicație și utilizate într-un scop global.
- ▶ **Obiectiv BDD: Administrarea și prelucrarea datelor distribuite**, situate pe diferite calculatoare și exploataate de sisteme eterogene.
- ▶ Bazele de date distribuite sunt sisteme de baze de date cooperante care rezidă pe mașini diferite, în locuri diferite.
- ▶ Această mulțime de baze de date este **exploatată de utilizator ca și cum ar fi o singură bază de date**.
- ▶ Programul de aplicație care exploatează o bază de date distribuită poate avea acces la date rezidente pe mai multe mașini, fără ca programatorul să cunoască localizarea datelor.
- ▶ Modelul **relațional** a rămas instrumentul principal prin care se realizează prelucrarea datelor distribuite.

EVOLUȚIA BAZELOR DE DATE

Baze de date cu suport decizional

- ▶ Sistemele informaticice, în particular bazele de date, au ajuns la maturitate.
- ▶ Cantitate mare de informații ale companiilor, păstrată în **tabele istorice**
 - ▶ nefolositoare sistemelor operaționale ale companiei, care funcționează cu **date curente**.
- ▶ **Analiza** date istorice → informații despre tendințe și evoluții care ar putea interesa compania.
 - ▶ Sunt necesare tehnologii și instrumente speciale.
 - ▶ Sunt analizate colecții de date provenind din sistemele operaționale ale companiei sau din surse externe.
- ▶ Principalul scop al acestor baze de date a fost de a întâmpina nevoile sistemelor operaționale, a căror natură este inherent tranzacțională.

EVOLUȚIA BAZELOR DE DATE

Baze de date cu suport decizional (continuare)

- ▶ **Sistemele tranzacționale** sunt interesate, în primul rând, să controleze la un moment dat **o singură tranzacție**.
- ▶ Un sistem operațional tipic operează cu evenimente predefinite și necesită acces rapid la date.
- ▶ Nevoile sistemelor operaționale nu se schimbă mult în timp.

EVOLUȚIA BAZELOR DE DATE

Baze de date cu suport decizional (continuare)

- ▶ S-au pus la punct principii și tehnologii noi care să servească procesului de **analiză** și **administrare** a datelor.
- ▶ O bază de date optimizată în acest scop definește o **Data Warehouse** (magazie de date)
- ▶ Principiul pe care îl urmează este cunoscut sub numele de procesare analitică (**OLAP – On Line Analytical Processing**).
- ▶ Principiul pe care se bazează sistemele tranzacționale a fost numit procesare tranzacțională (**OLTP – On Line Transactional Processing**).

EVOLUȚIA BAZELOR DE DATE

Baze de date cu suport decizional (continuare)

- ▶ Aplicațiile unei *Data Warehouse* trebuie să ofere răspunsuri unor întrebări de tipul:
 - ▶ „Care zi din săptămână este cea mai aglomerată?”
 - ▶ „Ce clienți fideli nu au beneficiat de reduceri de prețuri?“.
- ▶ **Interrogările** pe BD analitice sunt *ad-hoc*, nu sunt predefinite → 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

► OLTP versus Data Warehouse

Sistemele OLTP	Data Warehouse
Păstrează date curente	Păstrează date istorice
Stochează date detaliate	Stochează date detaliate, aggregate ușor sau puternic
Datele sunt dinamice	Datele sunt în mare măsură statice
Prelucrare repetitivă	Prelucrare ad-hoc, nestructurată și euristică
Nivel înalt de transfer al tranzacțiilor	Nivel mediu sau scăzut de transfer al tranzacțiilor
Tipar de utilizare previzibil	Tipar de utilizare imprevizibil
Conduse prin tranzacții	Conduse prin analiză
Susțin deciziile de zi cu zi	Susțin deciziile strategice
Deservesc un număr mare de utilizatori	Deservesc un număr relativ redus de utilizatori din administrație
Orientate spre aplicații	Orientate spre subiect

EVOLUȚIA BAZELOR DE DATE

Baze de date cu suport decizional (continuare)

- ▶ 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.
 - ▶ 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 și **optimiza** interogări de un tip special
 - ▶ posibilitatea de a folosi **mai multe procesoare** pentru a rezolva o interogare.

EVOLUȚIA BAZELOR DE DATE

Baze de date cu suport decizional (continuare)

- ▶ 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 mari cantități de date, a căror corelare nu este neapărat evidentă, în vederea descoperirii de tendințe și tipare.

Arhitectura multitier a sistemului Oracle

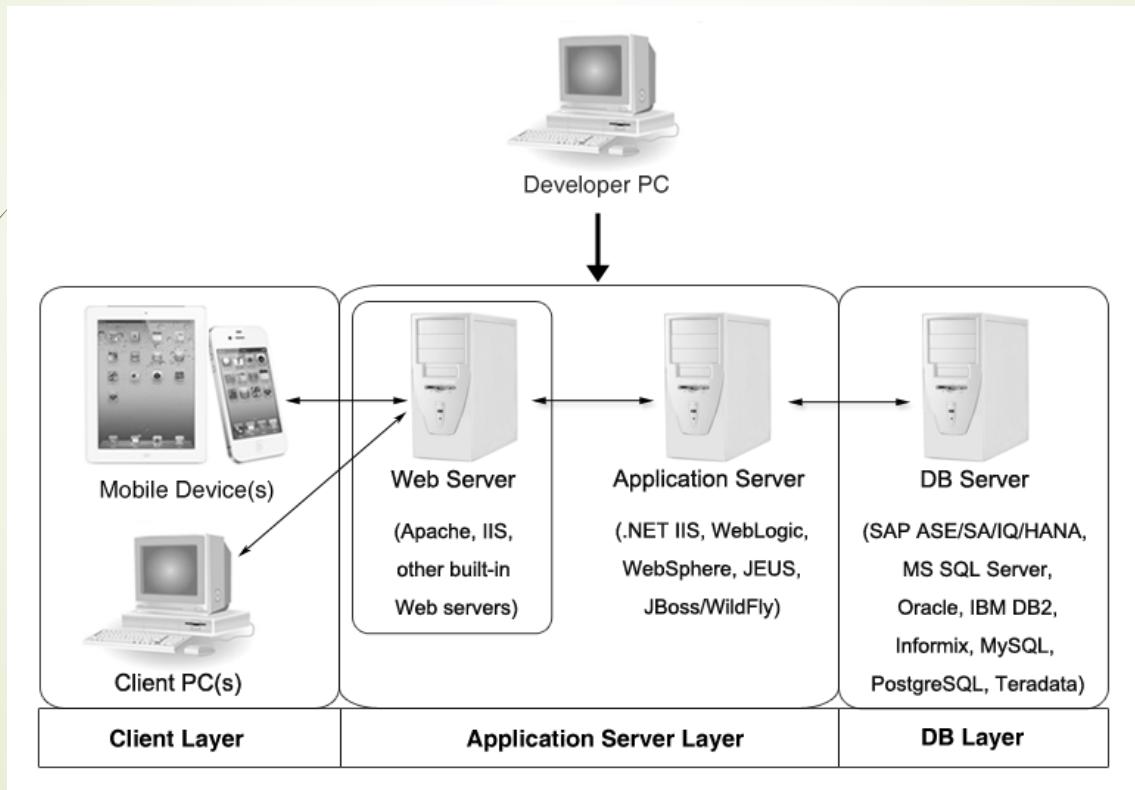
- ▶ **Arhitectura cu mai multe niveluri (*multitier*)** conține următoarele elemente:
 - ▶ unul sau mai mulți *clienti* care inițiază operații;
 - ▶ unul sau mai multe *servere* de aplicații care execută părți ale operațiilor;
 - ▶ un *server* de baze de date care stochează datele folosite de operații.
 - ▶ Privilegiile *server-ului* de aplicații sunt limitate pentru a preveni execuția operațiilor nedorite sau inutile în timpul unei operații *client*.

Arhitectura *multitier* a sistemului *Oracle*

- ▶ **Clientul** (un browser Web sau un proces *user*) → **cerere** pentru baza de date.
 - ▶ Conectarea la *server-ul* bazei de date se face printr-unul sau mai multe *server-e* de aplicații.
- ▶ **Server-ul de aplicații** constituie interfața dintre *clienti* și *serverul* bazei de date, asigurând **accesul la informații**. De asemenea, el include un nivel adițional pentru **securitate**.
- ▶ **Server-ul de baze de date** pune la dispoziția *server-ului* de aplicații informațiile necesare pentru soluționarea operațiilor lansate de către *client*.

Arhitectura *multitier* a sistemului Oracle

► Arhitectura *n-tier* :



https://docs.appeon.com/appeon_online_help/ps2019/installation_guide/ch37s02.html

BAZE DE DATE

CURS 2

Modelarea entitate-relație

PROIECTAREA BAZELOR DE DATE RELAȚIONALE

Modelarea entitate-
relație (E/R)

Diagramme entitate-
relație

Modelul relațional

Modelarea entitate-relație (E/R)

MODELAREA ENTITATE-RELATIE

Ce înțelegem prin model?

Model vs. Implementare?

Ce trebuie să cunoască utilizatorii?

MODELAREA ENTITATE-RELATIE

Model = reprezentare a **obiectelor** și **evenimentelor** lumii reale și a **asocierilor** dintre ele.

- abstractizare asupra aspectelor semnificative ale unei „întreprinderi”, ale unui sistem real

Model vs. Implementare?

- Caz particular al deosebirii uzuale dintre logic și fizic.

MODELAREA ENTITATE-RELATIE

- **3 tipuri fundamentale de modele, care descriu aspecte:**
 - **Statice**
 - **Dinamice**
 - **Funcționale**
- 
- ale procesului de **modelat**

MODELAREA ENTITATE-RELATIE

- Ce este un model de date?

MODELAREA ENTITATE-RELATIE

- **Model de date** = o colecție integrată de **concepte** necesare descrierii:
 - **datelor**,
 - **relațiilor** dintre ele,
 - **constrângerilor** existente asupra datelor sistemului real analizat.

MODELAREA ENTITATE-RELATIE

- Ce reprezintă **modelarea unei baze de date?**

MODELAREA ENTITATE-RELATIE

- **Modelarea unei baze de date** -> trecerea de la **percepția** unor fapte din lumea reală la reprezentarea lor prin **date**.
- **Modelul de date** trebuie:
 - să **reflecte** fidel fenomene ale lumii reale,
 - să **urmărească evoluția** acestei lumi și
 - să permită **comunicarea** dintre fenomenele lumii reale.

MODELAREA ENTITATE-RELATIE

- **Modelul de date** -> 3 componente :
 - o mulțime de **reguli** conform cărora sunt construite bazele de date (**partea structurală**);
 - o mulțime de **operații** permise asupra datelor, care sunt utilizate pentru **reactualizarea** sau **regăsirea** datelor (**partea de prelucrare**);
 - o mulțime de **reguli de integritate**, care asigură **coerența** datelor.

MODELAREA ENTITATE-RELATIE

- **Modelarea semantică a datelor** -> 4 etape:
 1. Se identifică o **multime de concepte semantice** care sunt utile în descrierea lumii reale.
 - Se presupune că lumea reală (modelul real analizat) este formată din **entități** care au anumite **proprietăți**, că fiecare entitate are o **identitate**, că există **legături**, corelații între entități. Conceptul de corelație, ca și cel de entitate, este util, în mod intuitiv, la descrierea modelului.
 2. Se caută o **multime de obiecte formale**, simbolice care sunt utilizate pentru reprezentarea conceptelor semantice anterioare.

MODELAREA ENTITATE-RELATIE

3. Se dă **reguli de integritate formale și generale** (constrângeri) care să reflecte restricțiile la care este supus modelul.
4. Se definește o **mulțime de operatori formali** prin care pot fi prelucrate și analizate obiectele formale.

Modelul entitate-relație

MODELUL ENTITATE-RELAȚIE

- P. Chen, 1976
- Abordare a modelării semantice
- Model de date conceptual, pentru a ușura proiectarea bazelor de date
- De nivel înalt, independent de platforma *hardware* utilizată și de tipul SGBD-ului
- Reprezentat grafic prin diagrame E/R

MODELUL ENTITATE-RELATIE

Baza de date -> mulțime de date ce **modelează** un sistem real format din:

- **Obiecte**
- **Legături între ele**

=> **Modelul E/R** împarte elementele unui sistem real în două categorii:

- **entități**
 - **relații** (legături, asocieri) între aceste entități.
-
- Entitățile și legăturile -> **characteristici (attribute)**.

MODELUL ENTITATE-RELATIE

- *Conceptul de relație, în sensul de asociere, care intervine în definirea diagramei E/R*

!=

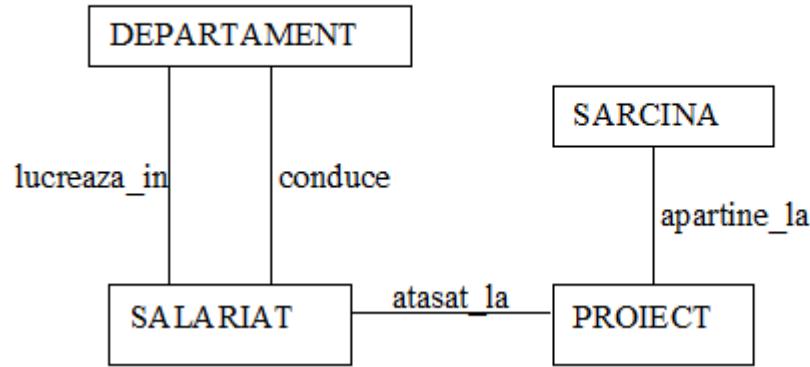
- *conceptul de relație care este specific modelului relațional.*

Diagrame entitate-relație

DIAGRAME ENTITATE-RELATIE

- **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).
- **Entitate**: persoană, loc, concept, activitate, eveniment care este semnificativ pentru ceea ce modelăm.

DIAGRAME ENTITATE-RELATIE



DIAGRAAME ENTITATE-RELATIE

- **Entitățile** vor deveni **tabele** în modelul relațional.
- 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 aceeași diagramă, două entități cu același nume, sau o aceeași entitate cu nume diferite.

DIAGRAME ENTITATE-RELATIE

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

DIAGRAME ENTITATE-RELATIE

- **Relație** (asociere): o **comunicare** între două sau mai multe entități. Existența unei relații este **subordonată existenței entităților** pe care le leagă.
 - 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 și minimă)** relației, adică numărul de tupluri ce aparțin relației.

DIAGRAME ENTITATE-RELATIE

poate (cardinalitate maximă) → **trebuie** (cardinalitate minimă)

- Câți salariați **pot** lucra într-un departament? Mulți!
 - În câte departamente **poate** lucra un salariat? În cel mult unul!
- ➔ Relația SALARIAT_lucreaza_in_DEPARTAMENT are cardinalitatea maximă **many-one** (n:1).
-
- 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).

DIAGRAME ENTITATE-RELATIE

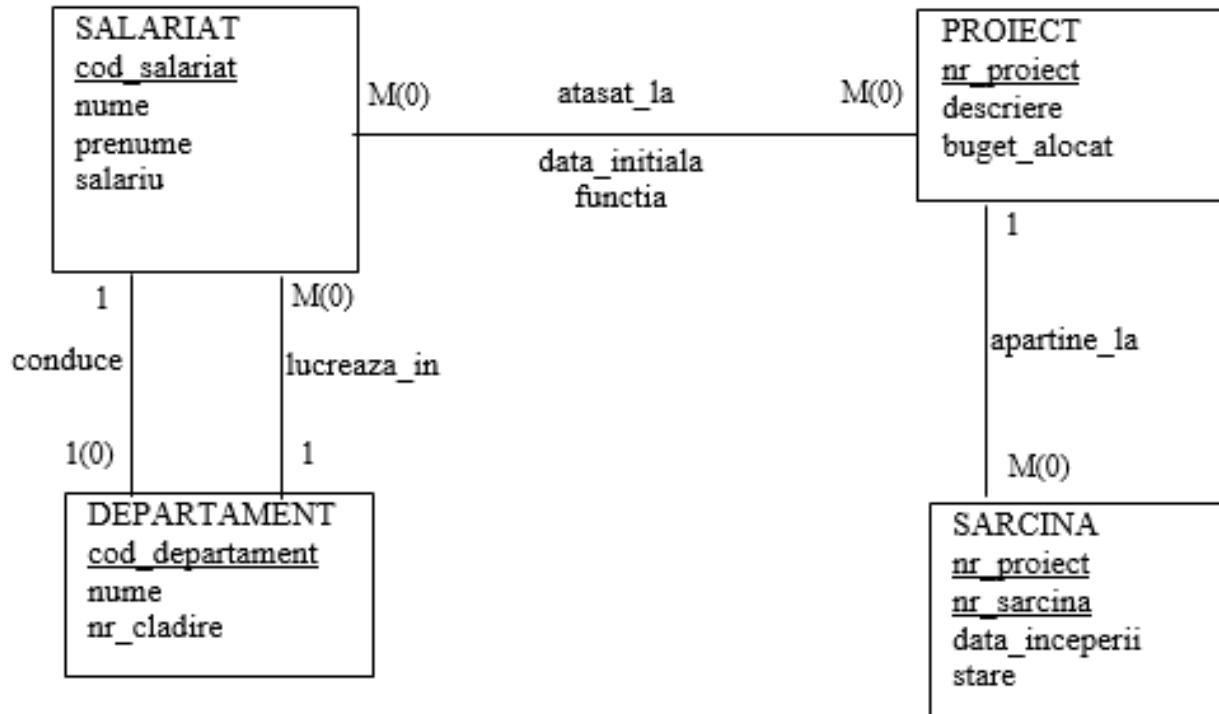
- **Atribut:** proprietate descriptivă a unei entități sau a unei relații.
 - Trebuie făcută distincția între **atribut** (devine coloană în modelele relationale) și **valoarea acestuia** (devine valoare în coloane).
 - Atributele sunt **substantive**, dar nu orice substantiv este atribut.
 - Fiecărui atribut trebuie să i se dea o **descriere** completă (exemplu, contraexemple, caracteristici).
 - Pentru fiecare atribut trebuie specificat **numele, tipul fizic** (*integer, float, char* etc.), **valori posibile, valori implicate, reguli de validare**

DIAGRAAME ENTITATE-RELATIE

Reguli (neunice) pentru proiectarea unei diagrame E/R:

- **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 sfârșitul numelui acestor atrbute;
- **cardinalitatea minimă** este indicată în paranteze, iar cardinalitatea **maximă** se scrie fără paranteze;
- nu este necesar să fie specificate, în cadrul diagramei, toate atrbutele.

DIAGRAME ENTITATE-RELATIE



DIAGRAME ENTITATE-RELATIE

Cazuri speciale de entități, relații, atribute

Dependență

Moștenirea atributelor

Specializare, generalizare

Relații recursive

Relații de tip 2, 3

Relație sau atribut?

Entitate sau relație?

Vezi curs și bibliografie!

DIAGRAME ENTITATE-RELATIE

Dependență

- **Entitate dependentă** – nu poate exista în mod independent (SARCINA depinde de PROIECT).
 - Cheia primară a unei entități dependente include cheia primară a sursei (*nr_project*) și cel puțin un atribut caracteristic entității (*nr_sarcina*).
 - (Entitatea dependentă se desenează prin dreptunghiuri cu linii mai subțiri.)

DIAGRAAME ENTITATE-RELATIE

Moștenirea atributelor
Specializare, generalizare

- **Moștenirea atributelor.**
 - **Subentitate** (subclasă) – submulțime a unei alte entități, numită **superentitate** (superclasă) (SALARIAȚ <—> 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, atrbutele și relațiile unei superentități sunt valabile pentru orice subentitate.** Afirmația reciprocă este falsă.

DIAGRAME ENTITATE-RELATIE

Moștenirea atributelor Specializare, generalizare

- **Generalizare.**

- Din entități similare care au mai multe **attribute comune** se pot crea **superentități**.
- Aceste superentități conțin attributele comune, iar attributele speciale sunt asignate la subentități. Pentru noile superentități se introduc **chei primare artificiale**.

- **Specializare.**

- După valorile unor attribute clasificatoare se pot determina **clase**.
- Un grup de **subentități reciproc exclusive** definește o clasă.
- (Clasele se aliniază în desen vertical.)

DIAGRAME ENTITATE-RELATIE

Relații recursive

- Într-o diagramă E/R se pot defini **relații recursive**.
- Exemplu?

DIAGRAME ENTITATE-RELATIE

Relații de tip 2, 3

- 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

DIAGRAME ENTITATE-RELATIE

- Trebuie **excluse** din model **relațiile indirecte** deoarece ele pot conduce la redundanță în baza de date.
- **Atributele derivabile** trebuie eliminate și introduse expresii prin care aceste atrbute pot fi calculate.
- Exemple?

DIAGRAME ENTITATE-RELATIE

Relație sau atribut?

- **Relație sau atribut?**
 - Dacă un atribut al unei entități reprezintă cheia primară a unei alte entități, atunci el referă o relație (*cod_departament* în tabelul SALARIAT).

Entitate sau relație?

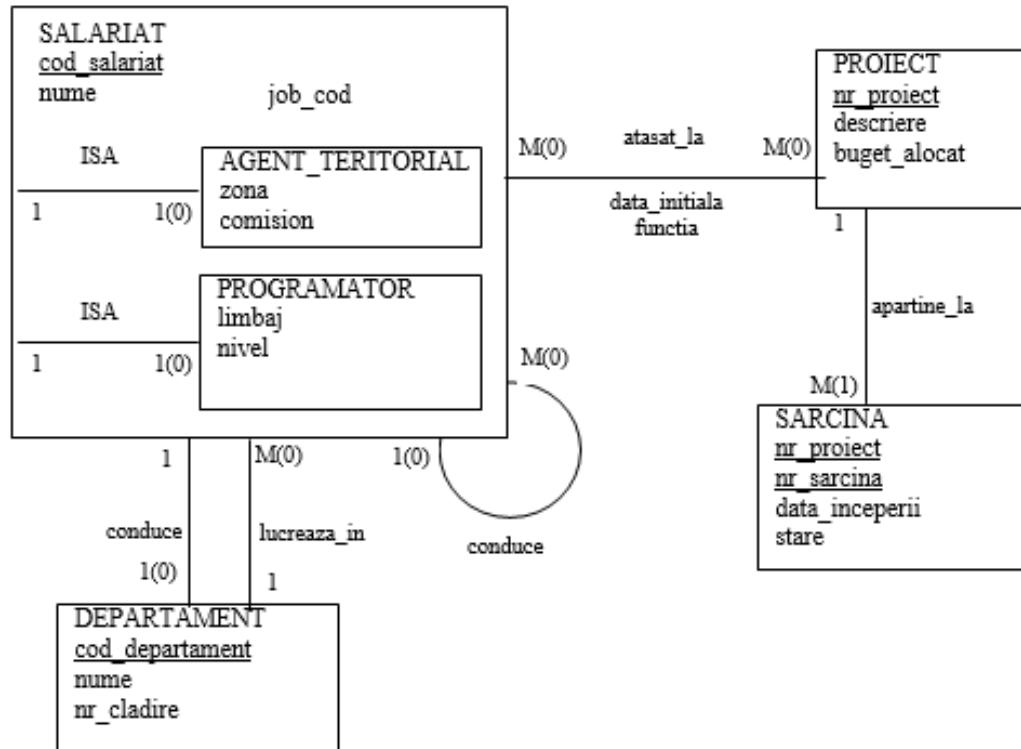
- **Entitate sau relație?**
 - Se cercetează cheia primară. Dacă aceasta combină cheile primare a două entități, atunci este vorba de o relație.
 - (cheia primară a relației *asociat_la* combină *cod_salariat* cu *nr_proiect*, prin urmare, *SALARIAT_asociat_la_PROJECT* va defini o relație și nu o entitate).

DIAGRAAME ENTITATE-RELATIE

Algoritmul pentru proiectarea diagramei E/R

1. identificarea **entităților** din cadrul sistemului analizat;
2. identificarea **relațiilor** (asocierilor) dintre entități și stabilirea **cardinalității**;
3. identificarea **atributelor** aferente entităților și asocierilor dintre entități;
4. stabilirea atributelor de identificare a entităților, adică stabilirea **cheilor primare**.

DIAGRAME ENTITATE-RELATIE



DIAGRAAME ENTITATE-RELATIE

- Aceeași realitate poate fi percepță diferit de către diversi analiști pentru un același sistem => **modele structurale distincte**.
 - Care sunt pașii următori?
 - Construirea **diagramei conceptuale**
 - obținerea **schemelor relaționale**
 - **normalizarea** acestora
- } =>
- => un **model relațional** care va elimina anumite **clase de anomalii** ce pot să apară în proiectarea modelului de date.

Modelul E/R extins

MODELUL E/R EXTINS

- Conceptele de bază ale modelării E/R nu sunt suficiente pentru a reprezenta cerințe complexe.
- Modelul E/R susținut cu concepte semantice adiționale definește **modelul E/R extins** (EER).
 - include toate **conceptele modelului original**
 - + conceptele adiționale de **subclasă, superclasă, moștenire, specializare, generalizare**.

MODELUL E/R EXTINS

- **Superclasa** (superentitatea) este o entitate care include subclase (subentități) distincte, ce trebuie reprezentate în modelul de date.
- **Subclasa** are un rol distinct și, evident, este membră a unei superclase. O subclasă, fiind o entitate, poate să posede propriile subclase.
 - O entitate împreună cu subclasele ei, subclasele acestora și aşa mai departe definește o **ierarhie de tip** (**ierarhie de specializare**). De exemplu, ANGAJAT_TEMP reprezintă o superclasă pentru entitatea MODEL.

MODELUL E/R EXTINS

- **Specializarea** este procesul de **maximizare a diferențelor** dintre membrii unei entități, prin identificarea caracteristicilor distinctive ale acestora.
 - Dacă subclasele unei specializări sunt **disjuncte**, atunci o entitate poate fi membră doar a unei subclase a acesteia (**constrângere de disjuncție**).
 - O specializare cu participare **totală** specifică faptul că fiecare entitate din superclasă trebuie să fie membră a unei subclase din specializare (**constrângere de participare**).
 - O specializare cu participare **parțială** specifică faptul că nu este necesar ca o entitate să aparțină vreunei subclase a acesteia. De exemplu, există salariați în PERS_CONTACT care nu aparțin niciunei subentități ale acesteia.

MODELUL E/R EXTINS

- **Generalizarea** este procesul de **minimizare a diferențelor** dintre entități, prin **identificarea caracteristicilor comune** ale acestora.
 - Generalizarea are ca rezultat identificarea unei superclase generalizate din subclasele initiale.

Deficiențe ale modelului E/R

DEFICIENȚE ALE MODELULUI E/R

- Cauza: interpretare eronata a sensului unei relații => **capcane de conectare**.
- Posibil sa necesite restructurarea modelului
- 2 clase de capcane de conectare:
 - **de intrerupere**
 - **in evantai**

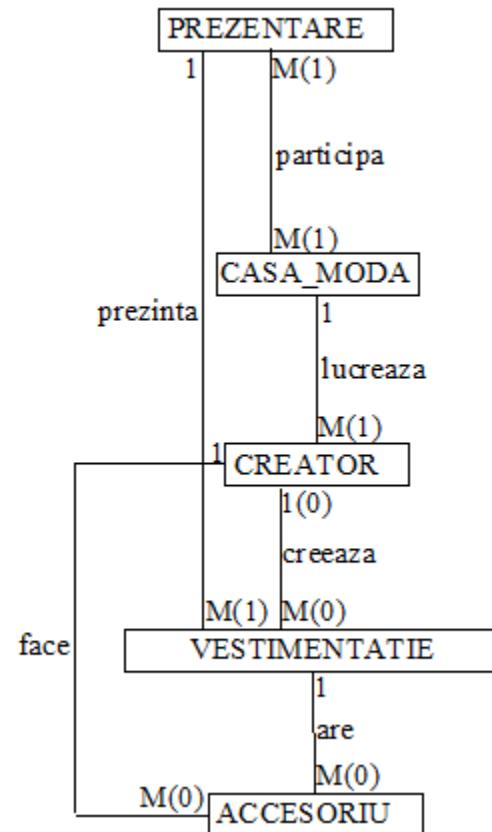
DEFICIENȚE ALE MODELULUI E/R

- **Capcane de întrerupere:** modelul sugerează existența unei relații între entități, dar **nu există o cale între anumite apariții ale entităților.**
 - Această capcană poate să apară acolo unde există o relație cu **participare parțială** (0 la cardinalitatea minimă), care face parte din calea dintre entitățile ce sunt legate.
- **Capcane în evantai:** modelul ia în considerare o relație între entități, dar **calea dintre anumite apariții ale entităților este ambiguă.**
 - Aceste capcane apar când **două sau mai multe relații one_to_many** provin din aceeași entitate.

DEFICIENȚE ALE MODELULUI E/R

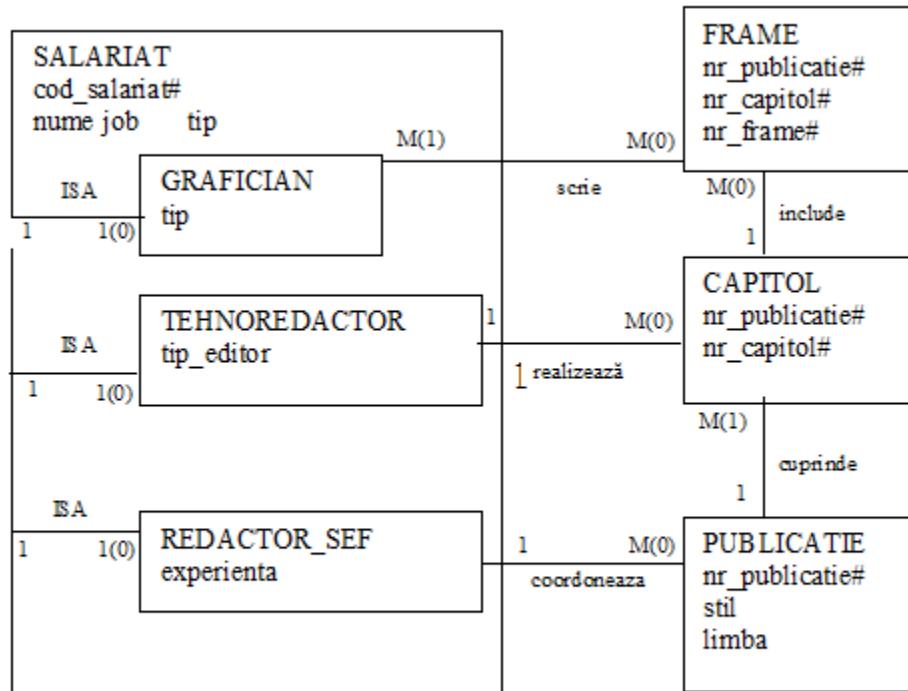
- Aceste capcane generează situațiile în care, aşa cum a fost proiectat modelul de date, **el nu poate să răspundă la anumite interogări.**
 - Dați un exemplu de capcană în evantai.
 - Exemplu de capcană de întrerupere: pentru a afla pentru ce prezentare de modă a fost creată o anumită vestimentație, a fost necesară introducerea unei legături între entitățile PREZENTARE și VESTIMENTATIE, care însă a generat redundanță în modelul de date:

DEFICIENȚE ALE MODELULUI E/R

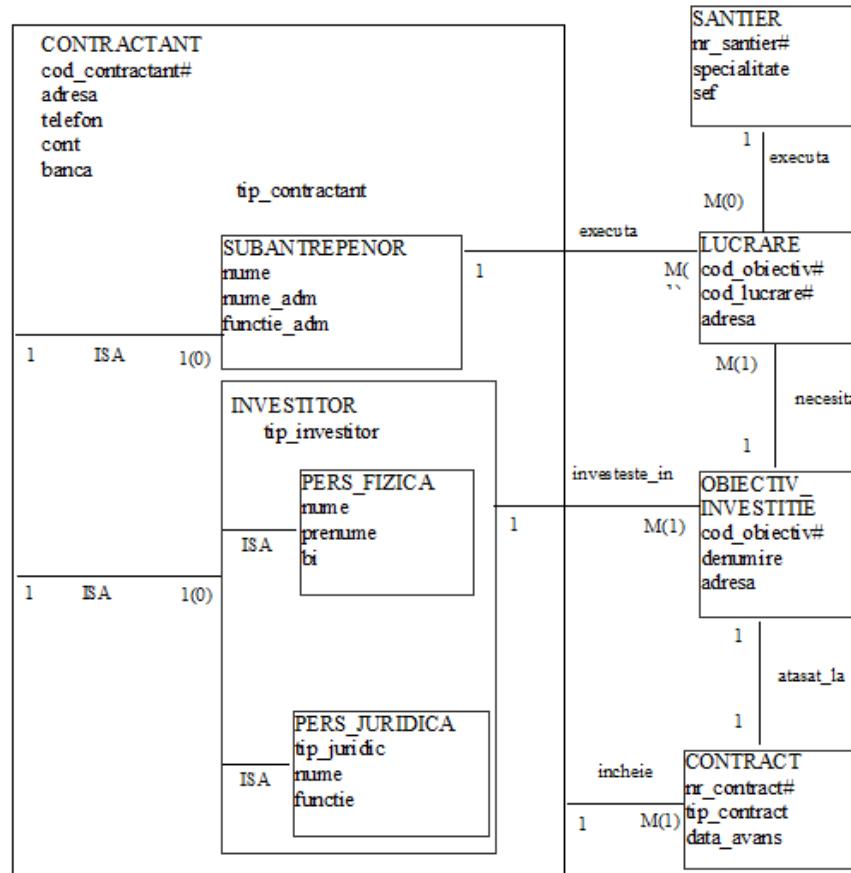


Exemple de diagramme E/R

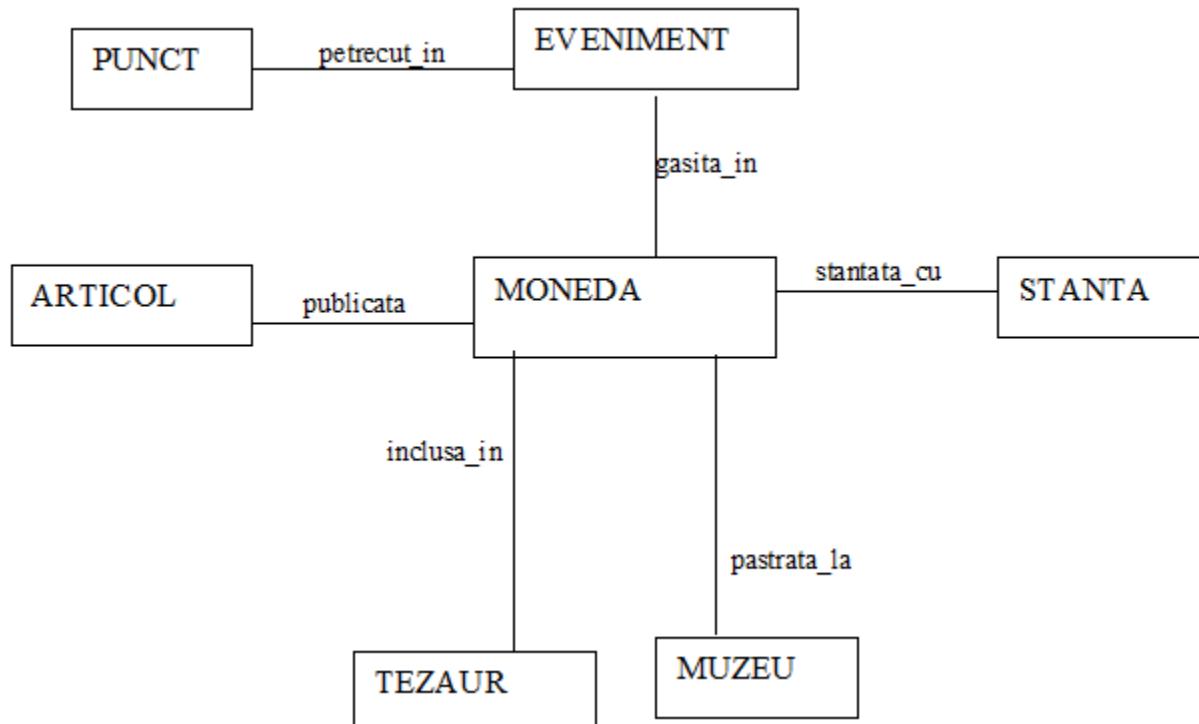
EXEMPLUL 1: Gestiunea Activităților de editare dintr-o editură



EXEMPLUL 2: Gestiunea unei firme de construcții



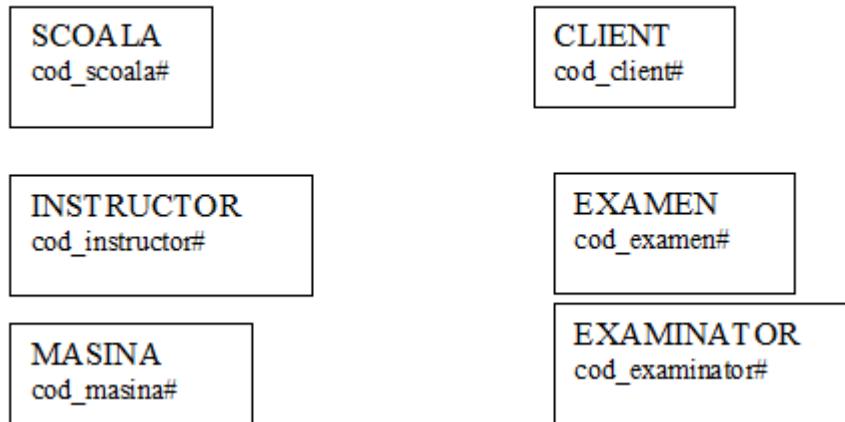
EXEMPLUL 3: Descoperiri de monede antice din România



Completați cardinalitatea!

STANȚA (nr_stanță, împărat emitent, valoare nominală, an emitere, monetăria, legenda de pe avers, legenda de pe revers) == > attribute ale entității **STANTA**

EXEMPLUL 4: Evidența școlilor de șoferi din România



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

EXEMPLUL 5: Campionatele de fotbal ale diferitelor țări

- Care este relația dintre entitățile MECI și ECHIPA? Ce cardinalitate are?

EXEMPLUL 6: Gestiunea activităților dintr-o agenție de turism

- Într-o agenție de turism lucrează ghizi, șoferi, agenți de vânzări.
- Din oferta agenției fac parte sejururi și excursii.
- Ghizii conduc excursii, la care sunt ataşați șoferi.
- Clienții agenției achiziționează sejururi sau excursii.
- Clienții sunt deserviți de către agenții de vânzări ai agenției.
- Un sejur se desfășoară într-o anumită locație.

TEMĂ

- Proiectați diagrama E/R pentru repartitia studentilor în căminele universității.

Bibliografie

- Popescu, I., Velcescu, L., ***Proiectarea bazelor de date***, Editura Universității din București, 2008 – Capitolul 2
- Connolly, T.M., Begg, C.E., Database Systems: ***A Practical Approach to Design, Implementation and Management***, 6th edition, Pearson Education, 2015 – Capitolele 11, 12

BAZE DE DATE

CURS 3

Modelarea entitate-relație (continuare).

Modelul relațional.

Observații temă (1/2)

1. Cardinalitățile se pun la ambele capete ale relației (nu doar la mijloc).
2. Relațiile diferite trebuie să fie semantic diferite. De exemplu: CAMIN_se_afla_in_LOCATIE, LOCATIE_are_CAMIN sunt o singură relație. Nu o dublăm în diagramă!
3. În anumite reprezentări, există "roluri" ale relației => regăsim verbe la ambele capete => OK (dar se recomandă notația de la curs).
4. Aceeași observație pentru alte reprezentări alternative, precum foot-crow. Nu o folosim, ci recurgem la reprezentarea cu cardinalitate maximă și minimă!
5. Nu punem attribute ce au valori multiple. Exemplu: FACULTATE nu are attribute precum specializari, profesori, CAMERA nu are lista_studenti, STUDENT nu are colegi etc.
6. Gardian, administrator, personal curătenie - o entitate cu atribut job și eventual subentități. **Când definim subentități?**
7. Greșeli la cardinalitate (valorile au fost inversate între capete).
8. Deoarece universitatea este sistemul al carui aspect îl modelez => NU este entitate în model! Ar fi fost dacă proiectam evidența căminelor din universitățile din România sau dintr-un oraș anume.
9. Unele modele au presupus (restrictiv) că un student poate urma o singură facultate și un cămin aparține în întregime unei singure facultăți. Regulile modelului trebuie să reflecte cât mai bine realitatea.

Observații temă (2/2)

10. Mai multe modele care au presupus că un student poate urma mai multe facultăți nu răspund la **întrebarea**: pe locurile cărei facultăți este cazat studentul?
11. Încă apare **CNP** drept PK – nu este o alegere adecvată din 2 motive: nu este ușor de folosit și reprezintă o informație confidențială!
12. OK pentru cei care au considerat "Camera" ca fiind dependentă de „Camin".
13. Au existat diagrame fără legătură între student și cameră – nu răspund la una dintre **întrebările de bază ale modelului**: cine este cazat într-o anumită cameră?
14. Nume cămin, nume facultate – NOK ca (parte din) cheia primară
15. Personalul (ca entitate) nu este dependent de cămin. Gândiți dependența ca o **relație de compunere**.
16. Studentul nu este subentitate a dosarului (sau este ales numele gresit pentru entitatea "Dosar"). Gândiți-vă dacă are sens propoziția <Entitate> este un / este o <Subentitate> !
17. Atribut locuri libere sau nr locuri? Mereu avem în vedere ce este invariabil și ce poate fi calculat pe baza altor elemente.
18. **Observație referitoare la dinamica modelului**: depinde de relevanța intervalului de timp pentru care proiectăm! Vezi exemplul din cursul anterior pasageri - autobuz vs student-camera.

Un alt exemplu: cel de la laborator. Este important acolo să știm istoricul angajatilor în companie dar nu avem un istoric al șefilor de departamente (daca nu cumva se poate deduce acest lucru tot din istoricul job-urilor)

EXEMPLUL 1

Gestiunea activităților de împrumut dintr-o bibliotecă

- **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 *cod_carte*.
 - **CITITOR** (entitate independentă) – orice cititor care poate împrumuta cărți. Cheia primară este atributul *cod_cititor*.
 - **DOMENIU** (entitate independentă) – domeniul căruia îi aparține o carte. Cheia primară este atributul *cod_domeniu*.
 - **IMPRUMUTA** – relație care leagă entitățile CITITOR și CARTE.
 - **APARTINE** – relație care leagă atrbutele CARTE și DOMENIU.
- Obs:** S-a presupus (restrictiv) că într-o zi un cititor nu poate împrumuta, de mai multe ori, aceeași carte -> regulă a modelului.
- Ce cardinalități au cele două relații?
 - Reprezentați diagrama E/R a acestui model.

EXEMPLUL 2

Gestiunea campionatelor de fotbal ale diferitelor țări

- **Entitățile** modelului sunt următoarele:
 - **ECHIPA, SPONSOR, MECI, ETAPA, CAMPIONAT**
-
- Precizați relațiile care există între aceste entități și cardinalitatea lor.
- Reprezentați diagrama E/R a acestui model.

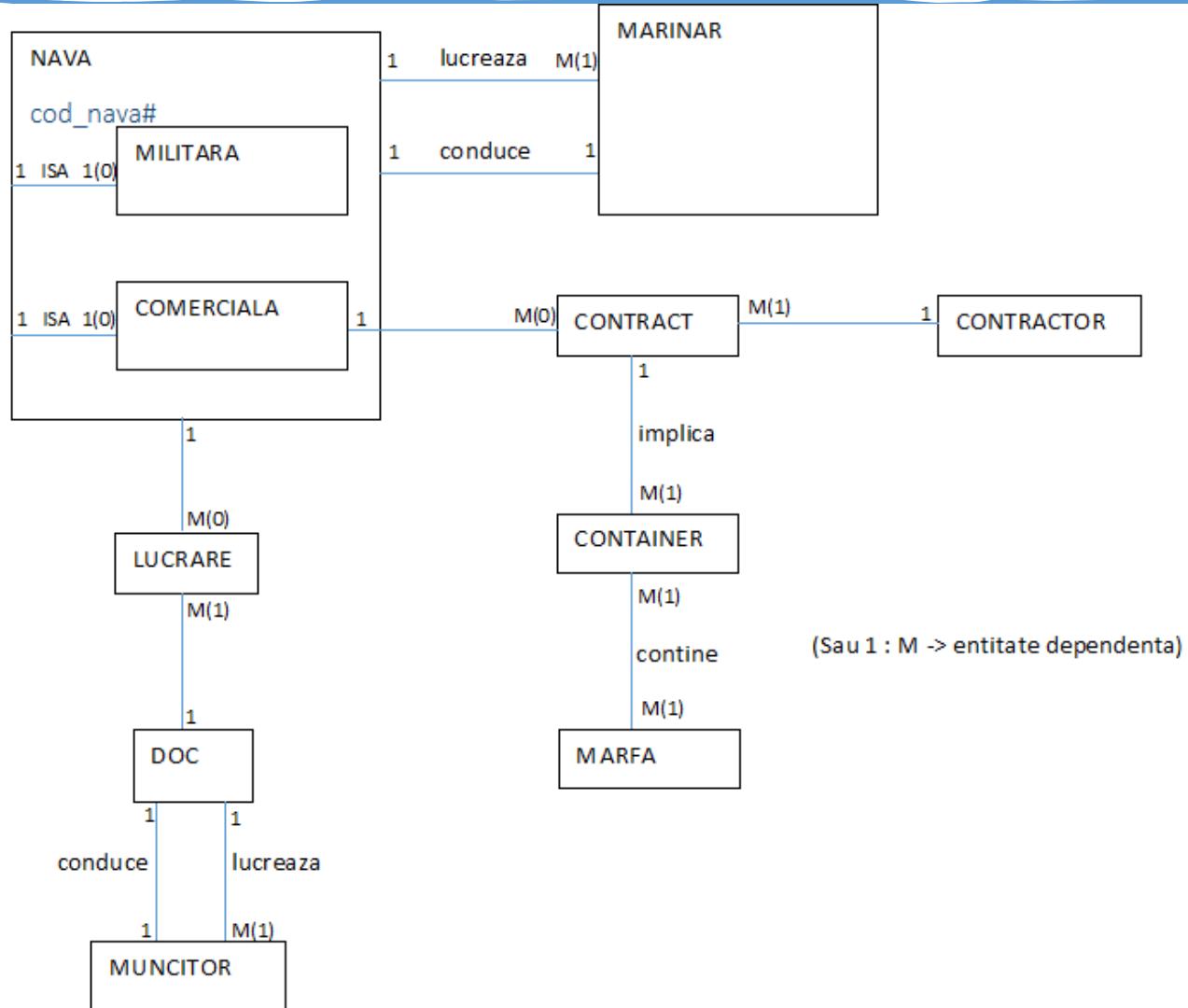
EXEMPLUL 3

Gestiunea activităților dintr-un port referitoare la servirea navelor

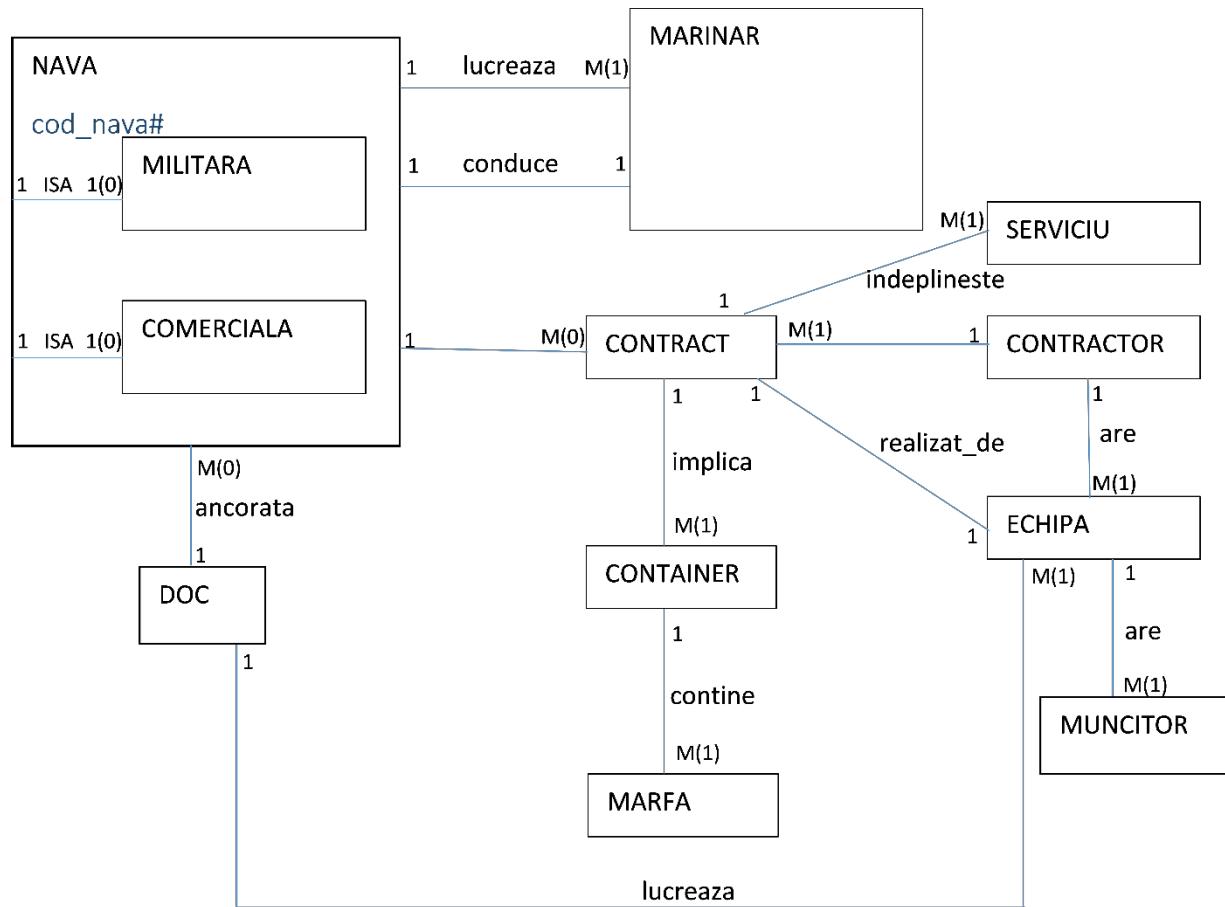
- Întrebări al căror răspuns trebuie să se regăsească în descrierea și regulile de funcționare ale modelului:
 - Ce tipuri de nave există?
 - Ce servicii pot fi oferite navelor (tipuri de lucrări)?
 - Ce atribute are entitatea DOC?
 - Ce tipuri de personal pot intra în componența diferitelor echipe?
 - Etc .
 - Exemple de specificații:
 - Pentru nave sunt semnate contracte de încărcare/descărcare cu contractori
 - Marfa ajunge în port în containere
 - Dați exemple de alte întrebări al căror răspuns trebuie specificat în descriere și reguli.
- Reprezentați diagrama E/R a unui model ce tratează cel puțin aspectele de mai sus.

EXEMPLUL 3 (continuare)

Care dintre variantele următoare este corectă? Corectați.



EXEMPLUL 3 (continuare)



PROIECTAREA BAZELOR DE DATE RELAȚIONALE



Modelarea entitate-
relație (E/R)



Diagramme entitate-
relație

Modelul relațional

Modelul relational

MODELUL RELAȚIONAL

- Concepții și dezvoltat de E.F. **Codd**
- 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 și prin urmare, orice interogare asupra bazei de date este tot o relație.**
- Cercetarea în domeniu → 3 mari proiecte (*System R, INGRES, PRTV*)

MODELUL RELAȚIONAL

- **Calități:**
 - este **simplu**;
 - **riguros** din punct de vedere matematic;
 - **nu este orientat** spre sistemul de calcul.
- Modalități pentru **definirea unui SGBD relațional**:
 - prezentarea datelor în **tabele** supuse anumitor operații de tip proiecție, selecție, reuniune, compunere, intersecție etc.
 - un sistem de baze de date ce suportă un limbaj de tip **SQL – Structured Query Language**;
 - un sistem de baze de date care respectă **principiile modelului relațional** introdus de E.F. Codd.

MODELUL RELAȚIONAL

- **Caracteristicile** unui model relațional:
 - **structura** relațională a datelor;
 - **operatorii** modelului relațional;
 - **regulile de integritate** care guvernează folosirea cheilor în model.

➤ Acste trei elemente corespund celor trei componente ale ingineriei *software*:

- **informație**
- **proces**
- **integritate.**

MODELUL RELAȚIONAL

Structura datelor

- Definirea noțiunilor de **domeniu**, **relație**, **schemă relațională**, **valoare null** și **tabel vizualizare (view)**.
- Conceptele utilizate pentru a descrie **formal**, **uzual** sau **fizic** elementele de bază ale organizării datelor:

Formal	Uzual	Fizic
Relație	Tabel / tabelă	Fișier
Tuplu	Linie	Înregistrare
Atribut	Coloană	Câmp
Domeniu	Tip de date	Tip de date

MODELUL RELAȚIONAL

- **Domeniu** – mulțime de valori care poate fi definită fie enumerând elementele componente, fie definind o proprietate distinctivă a domeniului valorilor.
- Fie D_1, D_2, \dots, D_n domenii finite, nu neapărat disjuncte. **Produsul cartezian** $D_1 \times D_2 \times \dots \times D_n$ al domeniilor D_1, D_2, \dots, D_n este definit de mulțimea **tuplurilor** (V_1, V_2, \dots, V_n) , unde $V_1 \in D_1, V_2 \in D_2, \dots, V_n \in D_n$. Numărul n definește **aritatea tuplului**.
- O **relație** R pe mulțimile D_1, D_2, \dots, D_n este o **submulțime** a produsului cartezian $D_1 \times D_2 \times \dots \times D_n$, deci este o **mulțime de tupluri**.
- Caracteristicile unei relații → comentat curs!

MODELUL RELAȚIONAL

- Definirea unei relații se referă la multimi care variază în timp.
- Este necesar 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 $R(A_1, A_2, \dots, A_n)$. Exemplu!
- Putem reprezenta o relație printr-un **tabel bidimensional**.
 - O coloană corespunde de fapt unui **atribut**.
 - Numărul atributelor definește **gradul** relației, iar numărul de tupluri din relație definește **cardinalitatea** relației.

MODELUL RELATIONAL

Exemplu (crearea unui tabel în SQL):

```
CREATE TABLE salariat (
    cod_salarat      NUMBER(4) PRIMARY KEY,
    nume              VARCHAR2(25),
    prenume            VARCHAR2(20),
    salariu           NUMBER(8, 2),
    sef                NUMBER(4),
    job_cod            VARCHAR2(6),
    cod_departament   NUMBER(3));
```

MODELUL RELAȚIONAL

Valoare *null*

- 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, și anume valoarea ***null***.
- Este necesară o aritmetică și 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 $\neg \text{null}$ este *null*.

MODELUL RELAȚIONAL

- Tabelele de adevăr pentru operatorii AND și OR:

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

MODELUL RELAȚIONAL

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.
 - **Securitate, reactualizări** → comentat la curs!
- 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 această vizualizare.

MODELUL RELAȚIONAL

Exemplu (crearea unei vizualizări în *SQL*):

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

MODELUL RELAȚIONAL

Reguli de integritate

→ **aserțiuni** 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

}

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**.

MODELUL RELAȚIONAL

- O **mulțime minimală** de atributе 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.
 - Atributеle care reprezintă cheia primară sunt fie subliniate, fie urmate de semnul # în diagrama E/R și în schemele relaționale.

MODELUL RELAȚIONAL

- 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 attribute din cadrul unei relații care conține o **cheie** a relației poartă numele de **supercheie**.
- Fie schemele relaționale $R1(P1, S1)$ și $R2(S1, S2)$, unde $P1$ este cheie primară pentru $R1$, $S1$ este cheie secundară pentru $R1$, iar $S1$ este cheie primară pentru $R2$. În acest caz, vom spune că $S1$ este **cheie externă** (cheie străină) pentru $R1$.

MODELUL RELAȚIONAL

Modelul relational respectă **trei reguli de integritate structurală**.

- **Regula 1 – unicitatea cheii.** Cheia primară trebuie să fie **unică** și **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

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).

PROIECTAREA MODELULUI RELAȚIONAL

Transformarea relațiilor

- Relațiile **1:1** și **1:n** devin **chei externe**.
 - Relația *conduce* devine coloană în tabelul DEPARTAMENT, iar relația *lucreaza_in* devine coloană în tabelul SALARIAT.
 - Simbolul „ \times “ indică plasamentul cheii externe, iar simbolul „ \underline{x} “ exprimă faptul că această cheie externă este conținută în cheia primară. Relația 1:1 plasează cheia externă în tabelul cu mai puține lini.
- 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.

PROIECTAREA MODELULUI RELAȚIONAL

Transformarea atributelor

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

PROIECTAREA MODELULUI RELAȚIONAL

- Cele patru tipuri de tabele (independente, dependente, subtabele și asociative) se deosebesc prin structura cheii primare.

Tabel	Reprezintă	Cheie primară
Independent	Entitate independentă	Nu conține chei externe
Subtabel	Subentitate	O cheie externă
Dependent	Entitate dependentă	O cheie externă și una sau mai multe coloane adiționale
	Atribut multiplu	
Asociativ	Relație m:n	Două sau mai multe chei externe și (optional) coloane adiționale
	Relație de tip 3	

- Diagramele conceptuale pentru proiectarea modelelor relaționale comentate vor fi construite din diagramele E/R prin adăugarea tabelelor associative și prin marcarea cheilor externe.

TEMĂ

Proiectați diagrama E/R și diagrama conceptuală pentru gestiunea unui club sportiv. Specificați schemele relaționale corespunzătoare diagramei conceptuale obținute.

Bibliografie

- Popescu, I., Velcescu, L., ***Proiectarea bazelor de date***, Editura Universității din București, 2008 – Capitolele 2, 3
- Connolly, T.M., Begg, C.E., Database Systems: ***A Practical Approach to Design, Implementation and Management***, 6th edition, Pearson Education, 2015 – Capitolele 16, 17

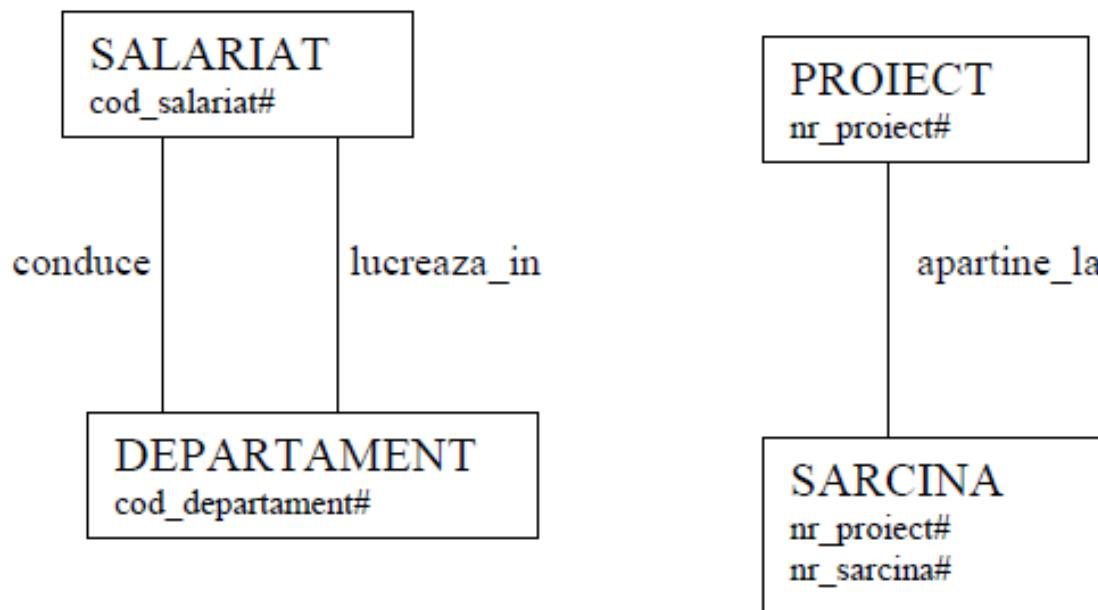
BAZE DE DATE

CURS 4

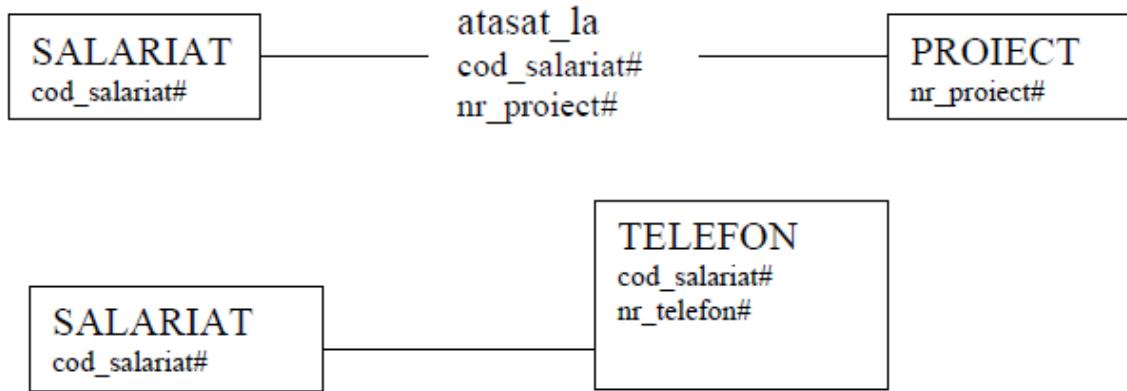
Modelul relațional (continuare).

DIAGRAME CONCEPTUALE

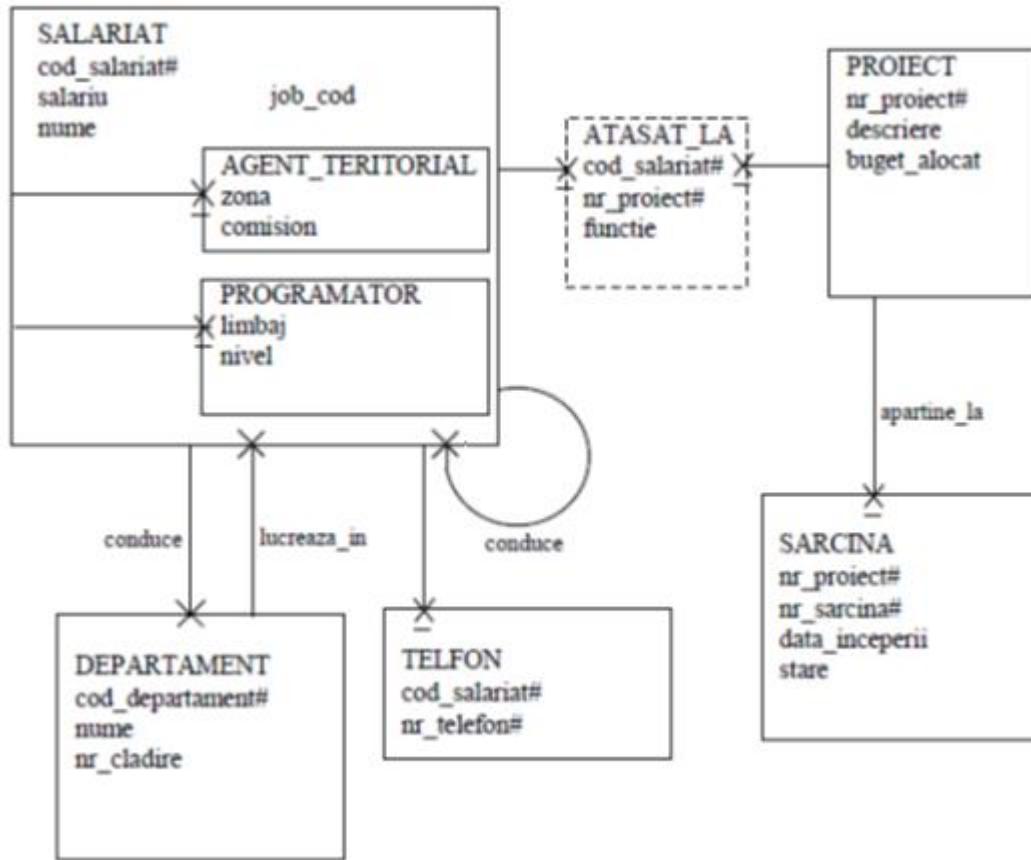
Completați următoarele fragmente de diagramă conceptuală:



DIAGRAME CONCEPTUALE



DIAGRAME CONCEPTUALE

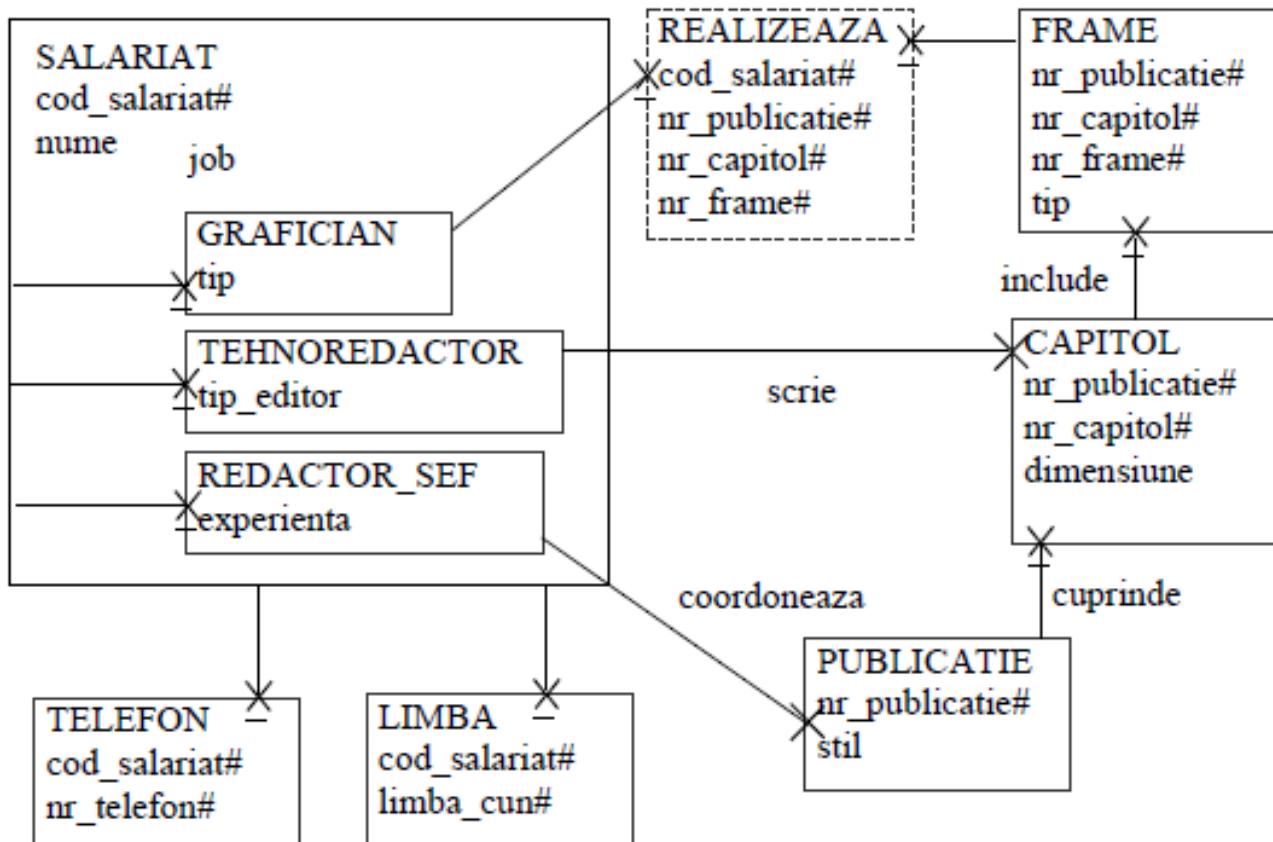


DIAGRAME CONCEPTUALE

Schemele relaționale corespunzătoare acestei diagrame conceptuale sunt următoarele:

- **SALARIAT**(cod_salariat#, nume, prenume, job_cod, cod_sef, forma_plata, nr_depart);
- **DEPARTAMENT**(cod_departament#, nume, numar_cladire, cod_sef);
- **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#).

DIAGRAME CONCEPTUALE



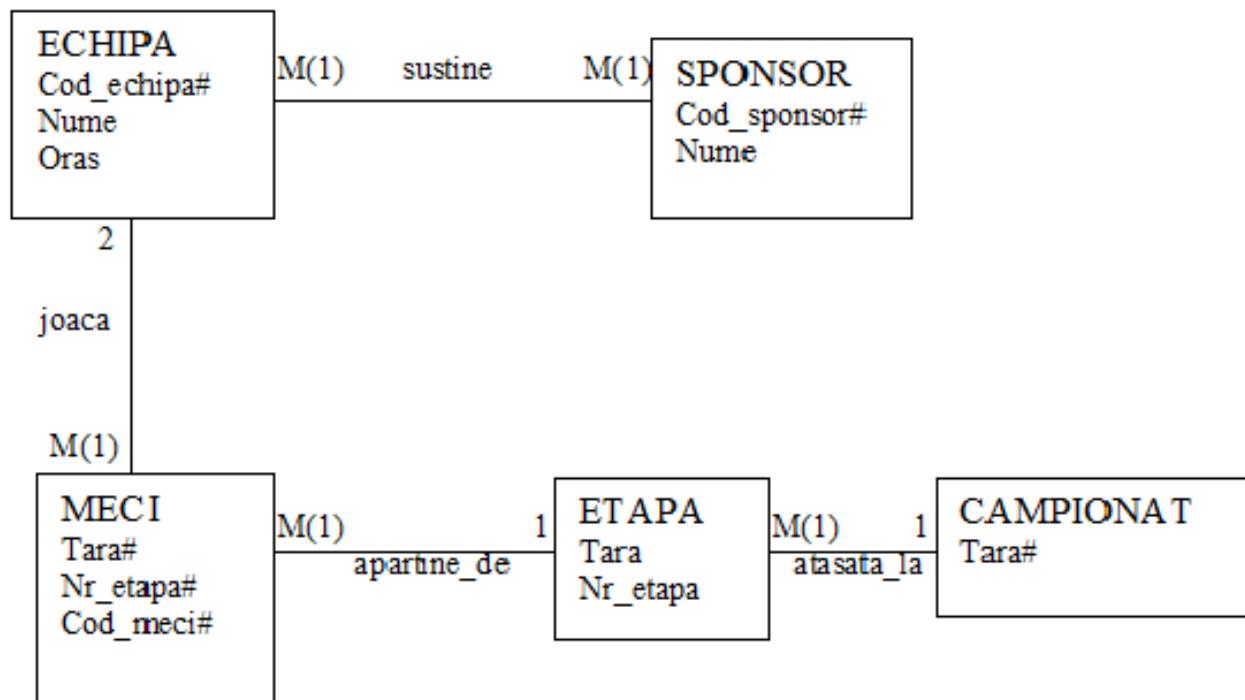
DIAGRAAME CONCEPTUALE

Schemele relationale:

- 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).

MODELUL RELAȚIONAL

Transformați următoarea diagramă E/R în diagramă conceptuală și enumerați schemele relaționale:



EXERCIȚII

1) Modelarea asocierilor existente între persoane și organizații.

- Persoanele activează în cadrul organizațiilor, unde îndeplinesc anumite roluri; o persoană poate avea roluri în mai multe organizații.
- Organizațiile și persoanele au adrese poștale și conturi de email.
- O adresă poate apartine mai multor persoane sau organizații.
- Între organizații există relații de diferite tipuri.
- Organizațiile pot avea prefixe și sufixe.

EXERCIȚII

2) Informații parțiale referitoare la gestiunea activităților unei societăți de asigurări.

- Societatea de asigurări are mai multe sucursale, în cadrul cărora lucrează asiguratorii. Între sucursale există relații de subordine.
- Clienții unei societăți de asigurări pot fi persoane fizice sau juridice.
- Clienții pot încheia contracte de tip CASCO sau locuință.
- Asigurarea pentru locuință prevede mai multe bunuri asigurate, iar cea CASCO include anumite dotări suplimentare.
- Pentru o poliță sunt efectuate plăți însotite de chitanțe.
- Un contract este încheiat cu un asigurator.

TEMĂ

Gestiunea cinematografelor din România: diagrama E/R, diagrama conceptuală, schemele relaționale.

BAZE DE DATE

CURS 5

Modelul relațional (continuare). Modelarea bazelor de date relationale utilizând UML.

Observații teme

1. CNP-ul este o bună cheie candidat, dar este mai bine să introducem o **CP artificială** fiindcă ramane, totuși, greu de referit (pe lângă faptul că este o dată cu caracter personal).
2. În schemele relaționale se pun și **cheile externe**.
3. Nu definim **subentități** dacă nu avem fie atribute specifice, fie participare la relații diferite. În acest caz, va fi suficient un atribut (de exemplu, tip_angajat).
4. Legat de întrebarea "ce tabel are **mai puține linii?**" pentru a ști unde se plasează cheia externă, răspunsul îl găsim în specificația modelului sau chiar în alte legături (de exemplu, în modelul HR, avem și o relație 1:M între DEPARTMENTS și EMPLOYEES => știm că vor fi mai puține linii în DEPARTMENTS). Totuși, uneori numărul de linii este (aproximativ) același. Unde adaugăm cheia externă? Se poate oriunde, dar în practică alegem tabelul în care este **mai important** să regăsim valoarea respectivă (de exemplu, este mai important să avem atribut "id_locatie" la cinematograf decât "id_cinematograf" la locație).

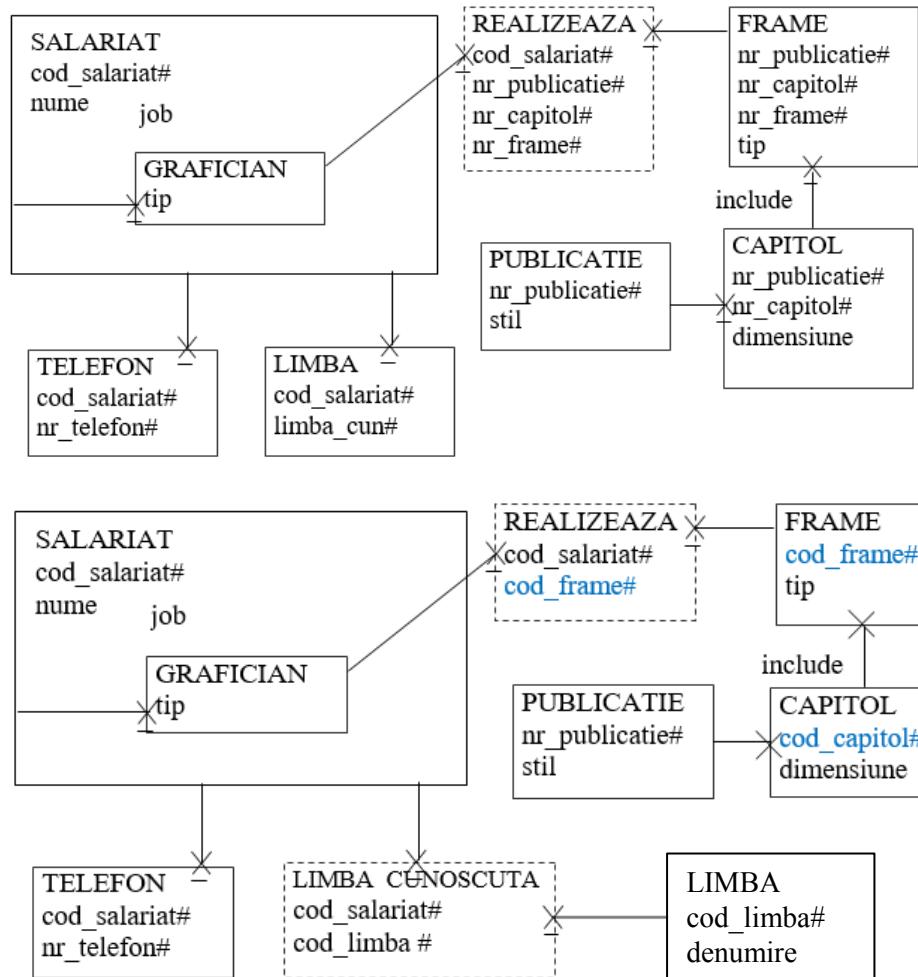
Completări design BD

1. Introducem o **CP artificială** dacă cheia primară este compusă din multe (în general, în practică, >2) atribute sau atunci când nu avem o cheie candidat suficient de simplă (vezi observație CNP).

Introducerea unei CP artificiale conduce la unele **modificări ale modelului obținut**:

- se pierde noțiunea de dependență!
 - se poate pune cheie artificială și pe tabelul asociativ, caz în care tabelul rămâne punctat în diagrama conceptuală, însă fără dependențe (“x” simplu, nu subliniat). Totuși, știm precis că CP a tabelului dependent nu va fi referită de vreo cheie externă, deci poate rămâne compusă.
-
2. Observație **attribute multiple**: alegem modelarea **ca tabel dependent sau relație M:M** (tabel asociativ)?
 - Dacă o aceeași valoare poate apartine mai multor entități, atunci poate fi o opțiune de design relația M:M.

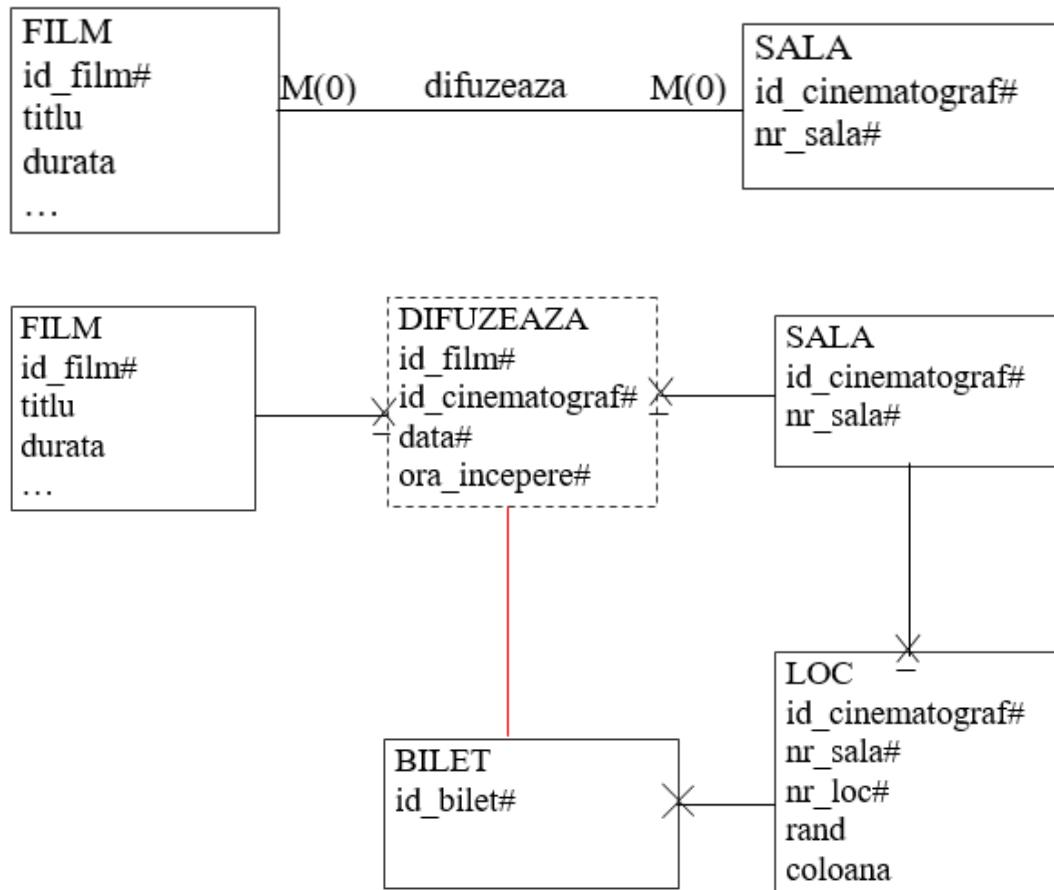
Completaři design BD



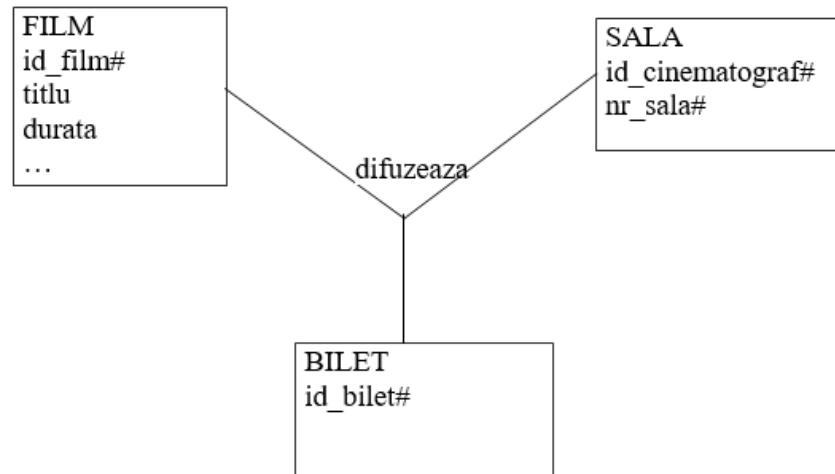
Completari design BD

3. Uneori ne aflăm în **situația în care "apar" legături cu tabele asociative** în diagrama conceptuală => pas înapoi în diagrama E/R pentru definirea corectă:
 - Definirea unei relații de tip 3
 - Considerarea tabelului asociativ ca entitate (pentru numele căreia găsim un substantiv potrivit)

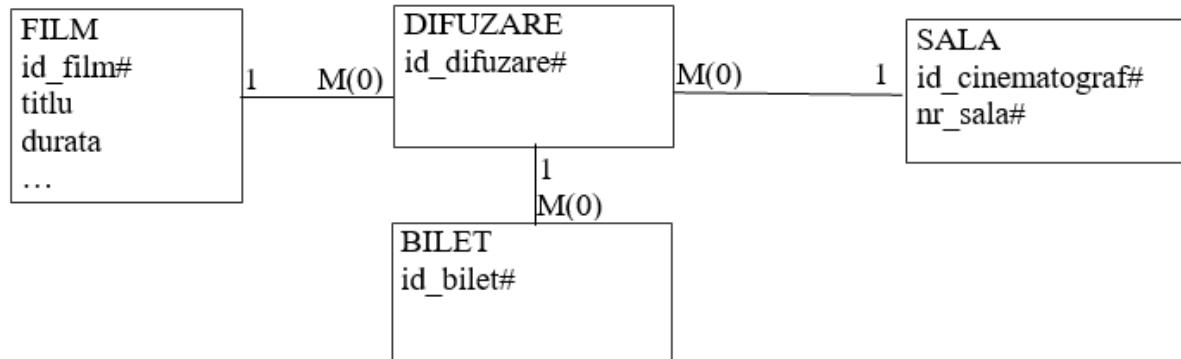
Completaři design BD



Completați design BD



Sau (mai bine, în acest caz – de ce?):



Modelarea bazelor de date relaționale utilizând UML

Modelare orientată pe obiecte cu UML

- Un **limbaj de modelare** reprezintă o modalitate de a **comunica** despre un sistem și de a **exprima** diversele modele produse în cadrul procesului de analiză și dezvoltare a sistemului.
 - Limbajul furnizează o **notație** care permite reprezentarea unui design.
- Ca orice limbaj, un limbaj de modelare are o **sintaxă** și o **semantică**.
 - Pentru semantica modelului trebuie aleasă **sintaxa cea mai expresivă**.
 - De cele mai multe ori, un limbaj de modelare este un **limbaj grafic** (diagramatic).

Modelare orientată pe obiecte cu UML

- *The Unified Modelling Language* (**UML**) este un limbaj **vizual** de **modelare** și de **comunicare** despre sistem.
- UML **nu este un limbaj de programare**.
- UML este un limbaj pentru modelarea orientată pe obiecte, cu suport pentru modelare vizuală, funcționând ca o modalitate de a comunica și de a exprima cunoștințe.

Modelare orientată pe obiecte cu UML

- **UML** este un **limbaj grafic de modelare** pentru **specificarea, vizualizarea, construcția și documentarea** componentelor:
 - unui sistem *software* (pentru întreprinderi, telecomunicații, sisteme bancare și financiare, transporturi etc.)
 - sau pentru modelarea organizațională a unor sisteme *non-software* (din justiție, medicină, afaceri etc.).
- **UML** permite realizarea tuturor **documentelor** necesare înțelegerii modelului și diagramelor utilizate pe tot parcursul ciclului de viață al unui proces de realizare a unui sistem.
 - specificarea **cerințelor, arhitecturii și proiectării** sistemului; elaborarea **codului** sursă, planuri de dezvoltare și de **management** al proiectului.

Modelare orientată pe obiecte cu UML

- UML **nu** este un limbaj de programare, dar modelele exprimate în UML pot fi implementate ușor în limbaje de programare orientate pe obiecte (C++, Java, C#) sau în **baze de date relaționale**.
 - Este posibilă nu numai generarea codului dintr-un model UML, dar și **ingineria inversă**, constând în construirea dintr-un cod dat a unui model UML.

Modelare orientată pe obiecte cu UML

- Sintaxa limbajului UML implică diagrame, în timp ce semantica lui se bazează pe paradigma orientării pe obiecte.
 - Din punct de vedere al modelării orientate pe obiecte, entitățile (concepțele) se numesc **clase**, iar relațiile dintre ele se numesc **asocieri**.
- UML conține mai multe tipuri de diagrame, fiecare reflectând unul sau mai multe dintre tipurile de vizualizări posibile asupra unui sistem *software*.

Modelare orientată pe obiecte cu UML

- **Diagrama claselor** descrie structura unui sistem în general. În componența ei intră **clase, stereotipuri și relațiile** dintre acestea.
- **Diagrama obiectelor** descrie **structura unui sistem** la un anumit moment. Acest tip de diagramă este o variantă a diagramei claselor care, în locul unei clase, prezintă mai multe instanțe ale ei, fiind formată din obiecte și legături dintre ele.

Modelare orientată pe obiecte cu UML

- **Diagrama cazurilor de utilizare** descrie **funcționalitatea** unui sistem, prezentând **actorii externi**, cazurile de utilizare identificate numai din punct de vedere al actorilor (**comportamentul sistemului**, aşa cum este perceput de utilizatorii lui), precum și **relațiile** dintre actori și cazurile de utilizare.
 - Un actor poate fi orice sau oricine interacționează cu sistemul (trimite sau recepționează mesaje de la sistem sau schimbă informații cu acesta). Actorul are un rol în cadrul unui sistem, nu este un utilizator individual al acestuia și, din acest motiv, el este o entitate (o clasă), nu o instanță. Un caz de utilizare este inițiat mereu de un actor și furnizează o valoare actorului.

Modelare orientată pe obiecte cu UML

- **Diagrama componentelor** (diagrama de implementare) descrie **structura fizică a codului** în termenii componentelor de cod și relațiilor dintre acestea.
- **Diagrama de desfășurare** (de exploatare) indică **arhitectura fizică pe care este implementat sistemul**, calculatoarele, nodurile sistemului și conexiunile dintre ele.
- **Diagrama secvențelor** este o **diagramă de interacțiune**, care prezintă colaborarea dinamică dintre un număr de obiecte, punând accentul pe secvențele de mesaje trimise între acestea pe măsura trecerii timpului.
- **Diagrama de colaborare** este tot o **diagramă de interacțiune**, dar care, pe lângă interacțiunea dintre obiecte (schimbul de mesaje), prezintă **obiectele și legăturile dintre ele**.

Modelare orientată pe obiecte cu UML

- **Diagrama de stare** descrie ciclul de viață al unui element (al obiectelor, subsistemelor și sistemelor), prin specificarea stărilor în care se găsește elementul și a evenimentelor care îi modifică starea.
- **Diagrama de activitate** prezintă activitățile și responsabilitățile elementelor sistemului, fiind utilizată pentru modelarea funcțiilor sistemului. Ea are ca elemente constitutive stări de acțiune și mesaje care vor fi trimise sau recepționate ca parte a acțiunii realizate.

Modelare orientată pe obiecte cu UML

În UML, diagramele fac parte din două categorii.

- **Diagrame dinamice** sau comportamentale – descriu **comportamentul** și **interacțiunile** dintre diverse entități ale sistemului informatic.
 - diagramele de secvență, colaborare, stare și activitate.
- **Diagrame statice** sau structurale - descriu **structura**, **responsabilitățile** sistemului informatic, **componentele** executabile ale sistemului, **locațiile fizice** de execuție și **nodurile de stocare** a datelor.
 - diagramele claselor, obiectelor, cazurilor de utilizare, componentelor și diagramele de exploatare.

Modelare orientată pe obiecte cu UML

- Diagramele UML pot fi desenate și administrate utilizând un utilitar **CASE** (*Computer Aided Software Engineering*).
 - Aceste utilitare sunt deosebit de utile în cazul unor diagrame complexe.
 - Totuși, dacă diagrama este prea complicată, atunci este necesară partiționarea ei în mai multe diagrame sau reprezentarea la un nivel superior de abstractizare.
- Exemple de utilitare CASE care permit realizarea diagramelor UML sunt reprezentate de: *Microsoft Office Visio*, *IBM Rational Rose Professional Data Modeler*, ***Lucidchart***, *Altova UModel*, *Borland Together*, *Visual Paradigm for UML*, *ArgoUML* etc.

Modelare orientată pe obiecte cu UML

- Interesul pentru suportul **modelării bazelor de date cu ajutorul UML** a condus la crearea unor **profiluri specifice**.
 - Un profil constituie **o propunere a unei comunități** și regrupează o mulțime de elemente UML care se aplică unui context particular și care conservă metamodelul UML.
 - *IBM Rational Rose* a inclus un astfel de profil adaptat bazelor de date.
- Există o **diferență între un model** (de exemplu, modelul conceptual de date) și un **formalism** (în care este descris un model).
- Astfel, putem vorbi despre **modelarea conceptuală a datelor urmând formalismul entitate-relație sau formalismul UML**. Notația exprimă doar aspectul referitor la reprezentare.

Modelare orientată pe obiecte cu UML

- Pe lângă **formalismul entitate-relație**, considerat **standardul de facto pentru modelarea datelor**, o opțiune alternativă este oferită de către UML.
- Aceasta include **primitive pentru modelarea datelor**, inițial concepute pentru reprezentarea structurii claselor unei aplicații orientate obiect, dar care pot fi folosite pentru **specificarea modelului de date al domeniului unei aplicații**.
 - În particular, **diagramele de clase UML** pot fi utilizate ca alternativă la **diagramele entitate-relație**.

Modelare orientată pe obiecte cu UML

- **Diferența majoră** dintre o diagramă de clase UML și o diagramă entitate-relație este reprezentată de **diferența dintre o clasă și o entitate**: o clasă este o generalizare a noțiunii de entitate, care permite proiectantului să specifice nu numai attribute, ci și funcții (numite metode) aplicabile instanțelor clasei.
- UML este **mai general** decât modelul entitate-relație
- iar proiectantul poate exploata diagramele de clase UML pentru a realiza **aceeași specificație** pe care o poate obține cu ajutorul modelului entitate-relație.

Modelare orientată pe obiecte cu UML

- Tabelul următor stabilește echivalențele dintre formalismele modelului entitate-relație și al notației UML:

Entitate-relație	UML
Tip entitate	Clasă
Asociere (relație)	Asociere (relație)
Entitate	Obiect
Cardinalitate	Multiplicitate
Model conceptual de date	Diagramă de clase

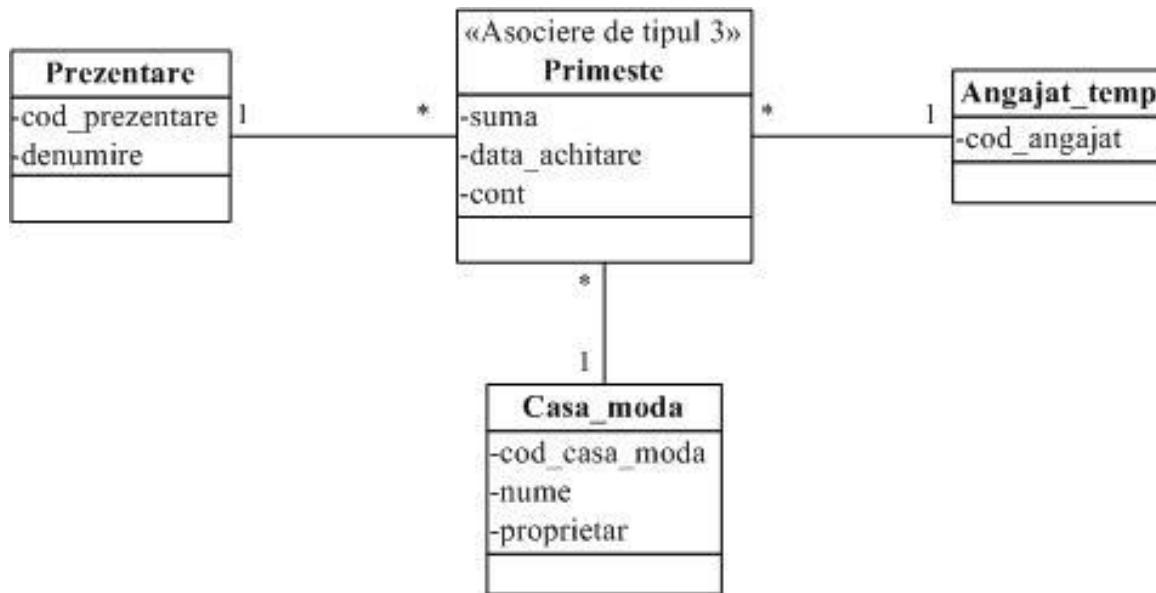
Modelare orientată pe obiecte cu UML

- Există **două forme de agregare**: compusă și partajată.
- **Compunerea** exprimă o relație de apartenență mai puternică, apărând atunci când relația este de tipul „compune” sau „**face parte din**”. Reprezentarea compunerii se face prin intermediul unui romb plin, poziționat lângă clasa care reprezintă întregul. Multiplicitatea extremității agregat nu poate depăși 1.
- **Agregarea** partajată presupune că un obiect **poate face parte din mai multe aggregate**, iar întregul se poate modifica în timp. O astfel de relație se reprezintă prin intermediul unui romb gol, poziționat lângă clasa care reprezintă întregul.

Modelare orientată pe obiecte cu UML

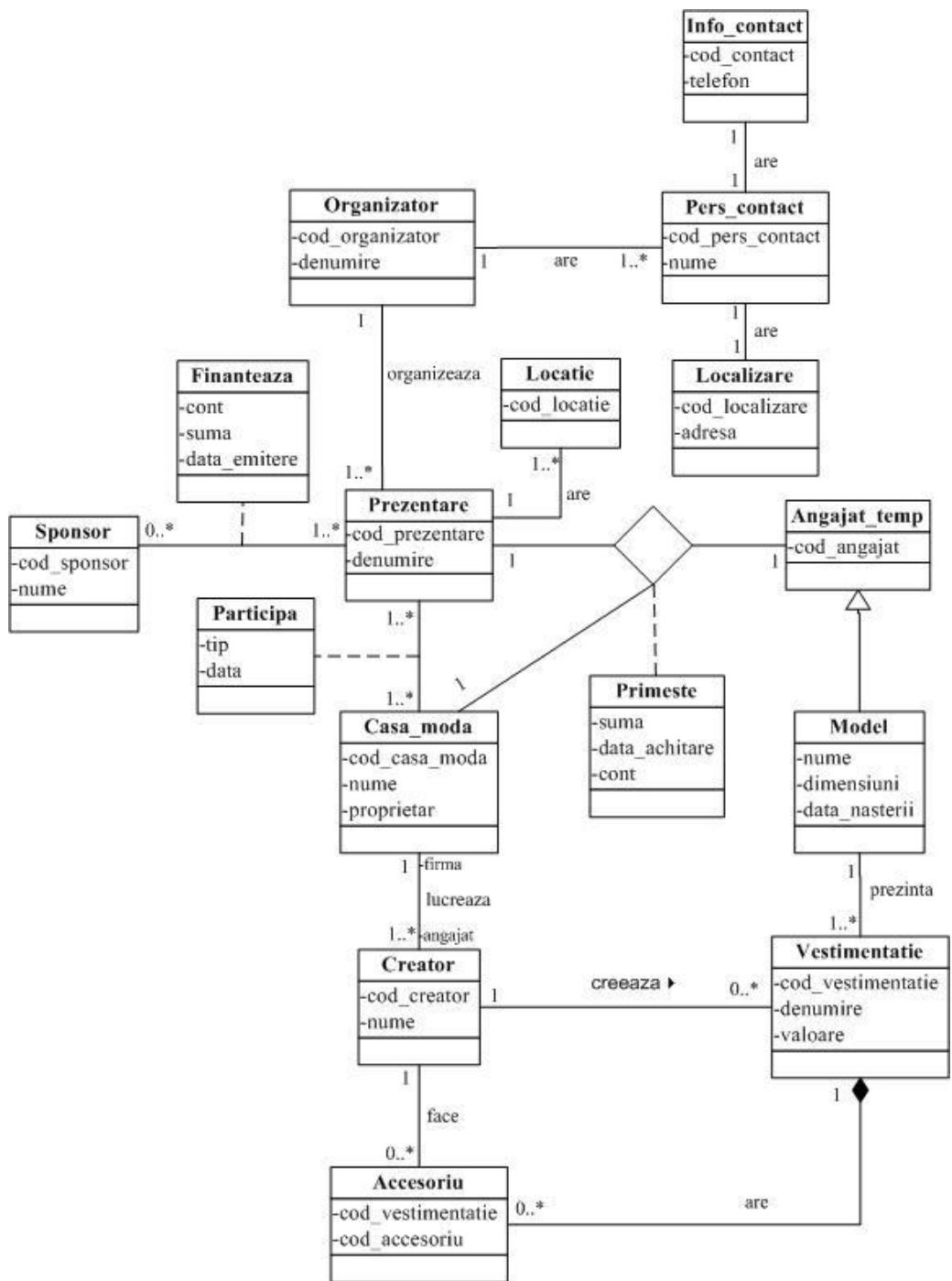
- Relațiile de grade strict mai mari decât 2 sunt reprezentate în UML cu ajutorul unui romb sau prin intermediul unei clase cu **stereotip**.
- Stereotipurile constituie **un mecanism de extensibilitate al UML**.
 - Ele permit extinderea vocabularului UML astfel încât să poată fi definite **noi elemente ale modelului**, derivate din cele existente dar cu proprietăți specifice domeniului problemei.
 - Reprezentarea grafică a unui stereotip se face prin plasarea numelui său, încadrat între caracterele „**<<** și **>>**” deasupra numelui unui alt element.

Modelare orientată pe obiecte cu UML



Asociere UML de tipul 3 cu stereotip.

Diagrama de clase corespunzătoare unei restrictii a modelului.



TEMA



- Assignment pe MS Teams.

BAZE DE DATE

CURS 6
Algebra relațională

ALGEBRA RELAȚIONALĂ

- **LDD** – precizează:
 - **entitățile**, relațiile dintre ele;
 - **attributele**, structura atributelor;
 - **cheile, constrângerile**.
- **LMD** – cuprinde aspecte referitoare la:
 - introducerea
 - eliminarea
 - modificarea și
 - căutarea datelor.

ALGEBRA RELAȚIONALĂ

- 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).
-
- **Echivalența** dintre **algebra relațională** și **calculul relațional** a fost demonstrată de J.D.Ullman.
 - Această echivalență arată că orice relație posibil de definit în algebra relațională poate fi definită și în cadrul calcului relațional, și reciproc.

ALGEBRA RELAȚIONALĂ

- **Algebra relațională** a fost introdusă de E.F. Codd
 - **mulțime de operații formale** acționând asupra unor relații și având ca rezultat alte relații.
- Baza teoretică pentru limbajele de interogare relaționale o constituie operatorii introdusi de Codd pentru prelucrarea relațiilor.
- **Operatorii sunt numai pentru citire** (nu actualizează operanzi)!!!

ALGEBRA RELAȚIONALĂ

- Scopul fundamental al algebrei relaționale este de a permite scrierea **expresiilor relaționale**.
 - reprezentare de nivel superior, simbolică, a intențiilor utilizatorului
 - pot fi supuse unei diversități de **reguli de transformare (optimizare)**.
- Relațiile sunt **închise** față de algebra relațională
 - operanții și rezultatele sunt **relații** → ieșirea unei operații poate deveni intrare pentru alta → posibilitatea **imbricării** expresiilor în algebra relațională.

ALGEBRA RELAȚIONALĂ

- **Operatorii algebrei relaționale** sunt:
 - operatori **tradiționali** pe mulțimi (**UNION**, **INTERSECT**, **PRODUCT**, **DIFFERENCE**);
 - operatori **relaționali** speciali (**PROJECT**, **SELECT**, **JOIN**, **DIVISION**).

ALGEBRA RELAȚIONALĂ

Operatorul PROJECT

- Proiecția este o operație unară care elimină anumite attribute ale unei relații producând o submulțime „pe verticală” a acesteia.
 - Suprimarea unor attribute poate avea ca efect apariția unor **tupluri duplicate**, care trebuie eliminate.
- **Notății:**
 - $\Pi_{A_1, \dots, A_m}(R)$
 - **PROJECT**(R, A_1, \dots, A_m)
 - $R[A_1, \dots, A_m]$

unde A_1, A_2, \dots, A_m sunt parametrii proiecției relativ la relația R .

ALGEBRA RELAȚIONALĂ

Exemplu. Să se obțină o listă ce conține numele, prenumele și job-ul angajaților.

1. Proiecție în algebra relațională:

Rezultat = **PROJECT** (SALARIAT, nume, prenume, job)

2. Proiecție cu dubluri în SQL:

```
SELECT nume, prenume, job
```

```
FROM salariat;
```

3. Proiecție fără dubluri în SQL:

```
SELECT DISTINCT nume, prenume, job
```

```
FROM salariat;
```

ALGEBRA RELAȚIONALĂ

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.
- **Notări:**
 - $\sigma_{\text{condiție}}(R)$
 - $R[\text{condiție}]$
 - **SELECT(R , condiție)**
 - **RESTRICT(R , condiție).**

ALGEBRA RELAȚIONALĂ

Exemplu. Să se obțină informații complete despre programatori.

1. Selecție în algebra relațională:

Rezultat = **SELECT** (SALARIAT, job = 'programator')

2. Selecție în SQL:

```
SELECT *
FROM   salariat
WHERE  job = 'programator';
```

ALGEBRA RELAȚIONALĂ

Operatorul UNION

- Reuniunea a două relații R și S este mulțimea tuplurilor aparținând fie lui R , fie lui S , fie ambelor relații.
- **Notății:**
 - $R \cup S$
 - **UNION(R, S)**
 - $OR(R, S)$
 - $APPEND(R, S)$.

Exemplu. Să se obțină lista cu numele persoanelor fizice și ale subantreprenorilor.

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

ALGEBRA RELAȚIONALĂ

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.
- **Notății:**
 - $R - S$
 - **DIFFERENCE(R, S)**
 - **REMOVE(R, S)**
 - **MINUS(R, S)**.

ALGEBRA RELAȚIONALĂ

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

1. Diferență în algebra relațională:

$R = \text{PROJECT} (\text{SELECT} (\text{CONTRACT}, \text{tip_contract}='T'), \text{nr_contract}, \text{tip_contract}, \text{val_investitie}, \text{durata_lucrare});$

$S = \text{PROJECT} (\text{SELECT} (\text{CONTRACT}, \text{val_investitie} > 60000), \text{nr_contract}, \text{tip_contract}, \text{val_investitie}, \text{durata_lucrare});$

Rezultat = **DIFFERENCE** (R, S)

ALGEBRA RELAȚIONALĂ

2. Diferență în SQL:

```
SELECT      nr_contract, tip_contract,  
            val_investitie, durata_lucrare  
FROM        contract  
WHERE       tip_contract = 'T'  
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 orașele în care se află cel puțin o filială, dar nicio proprietate.

ALGEBRA RELAȚIONALĂ

Operatorul INTERSECT

- Intersecția a două relații R și S este mulțimea tuplurilor care aparțin atât lui R , cât și lui S . Operatorul INTERSECT este un operator binar, comutativ, derivat:
 - $R \cap S = R - (R - S)$
 - $R \cap S = S - (S - R)$.
- Notații:
 - **INTERSECT(R, S)**
 - $R \cap S$
 - **AND(R, S)**.

ALGEBRA RELAȚIONALĂ

Operatorul INTERSECT (cont.)

- În anumite dialecte SQL există operator special (**INTERSECT**), care realizează această operație.
- Operatorii INTERSECT și DIFFERENCE pot fi simulați în SQL (în cadrul comenzi SELECT) cu ajutorul operatorilor EXISTS, NOT EXISTS, IN, != ANY.

ALGEBRA RELAȚIONALĂ

Exemplu. Utilizând tabelele *angajati* și *departamente*, să se obțină lista codurilor salariaților care sunt directori de departament, dar și șefi direcți ai altor persoane.

1. Intersecție în algebra relațională:

$R = \text{PROJECT} (\text{ANGAJATI}, \text{cod_sef})$;

$S = \text{PROJECT} (\text{DEPARTAMENTE}, \text{cod_director})$,

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

2. Intersecție în SQL:

```
SELECT    cod_sef  
FROM      angajati  
INTERSECT  
SELECT    cod_director  
FROM      departamente;
```

ALGEBRA RELAȚIONALĂ

3. Simularea intersecției în SQL:

```
SELECT cod_director  
FROM departamente d  
WHERE EXISTS  
(SELECT cod_sef  
FROM angajati a  
WHERE d.cod_director=a.cod_sef);
```



- Cum simulăm intersecția utilizând operatorul IN?
- Cum simulăm diferența de la exercițiul de pe slide-ul 14?

ALGEBRA RELAȚIONALĂ

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 .
- Notări:
 - $R \times S$
 - **PRODUCT(R, S)**
 - **TIMES(R, S)**.

ALGEBRA RELAȚIONALĂ

Exemplu. Să se obțină lista tuturor posibilităților de investiție în diverse obiective de către o firmă care este persoană juridică.

1. Produs cartezian în algebra relațională:

$R = \text{PROJECT}(\text{PERS_JURIDICA}, \text{nume}, \text{cod_contractant});$

$S = \text{PROJECT}(\text{OBIECTIV_INVESTITIE}, \text{denumire});$

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

2. Produs cartezian în SQL:

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

ALGEBRA RELAȚIONALĂ

Operatorul DIVISION

- Diviziunea este o operație binară care definește o relație ce conține valorile atributelor dintr-o relație care apar **în toate** valorile atributelor din cealaltă relație.
- **Notății:**
 - $\text{DIVIDE}(R, S)$
 - $\text{DIVISION}(R, S)$
 - $R \div S$.
- Diviziunea conține acele tupluri de dimensiune $n - m$ la care, adăugând **orice tuplu din S** , se obține un tuplu din R .
- Operatorul diviziune poate fi exprimat formal astfel:

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

ALGEBRA RELAȚIONALĂ

- Operatorul DIVISION este legat de cuantificatorul universal (\forall) care nu există în SQL.
- Cuantificatorul universal poate fi însă simulață cu ajutorul cuantificatorului existențial (\exists) 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.

ALGEBRA RELAȚIONALĂ

Exemplu.

$R(A, B, C, D); S(C, D)$

A	B	C	D
a	b	x	y
a	b	z	t
a	c	x	y
c	a	x	t
c	d	x	y
c	d	z	t

C	D
x	y
z	t

DIVISION(R, S) = ?

ALGEBRA RELAȚIONALĂ

Exemplu. Să se obțină codurile salariaților atașați **tuturor** proiectelor pentru care s-a alocat un buget egal cu 1000.

1. Diviziune în algebra relațională:

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

2. Diviziune în SQL:

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

ALGEBRA RELAȚIONALĂ

3. 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);
```

ALGEBRA RELAȚIONALĂ

Operatorul JOIN

- Operatorul de compunere permite regăsirea informației din mai multe relații corelate.
- Operatorul combină produsul cartezian, selecția și proiecția.
- 4 tipuri de join:
 - NATURAL JOIN
 - θ-JOIN
 - SEMI-JOIN
 - OUTER JOIN

ALGEBRA RELAȚIONALĂ

NATURAL JOIN

- Operatorul de compunere naturală (NATURAL JOIN) combină tupluri din două relații R și 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 și o coloană în S , se selectează din $R \times S$ tuplurile ale căror valori coincid în coloanele $R.A$ și $S.A$ (atributul $R.A$ reprezintă numele coloanei din $R \times S$ corespunzătoare coloanei A din R);
 3. pentru fiecare astfel de atribut A se elimină coloana $S.A$, iar coloana $R.A$ se va numi A .

ALGEBRA RELAȚIONALĂ

NATURAL JOIN (cont.)

- Operatorul NATURAL JOIN poate fi exprimat formal astfel:

$$\text{JOIN}(R, S) = \Pi_{i_1, \dots, i_m} \sigma_{(R.A_1 = S.A_1) \wedge \dots \wedge (R.A_k = S.A_k)}(R \times S),$$

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

ALGEBRA RELAȚIONALĂ

Exemplu. Să se obțină informații complete despre angajați și departamentele în care lucrează.

1. Operatorul de compunere naturală în algebra relațională:

Rezultat = **JOIN** (SALARIAT, DEPARTAMENT).

2. Operatorul de compunere naturală în SQL:

```
SELECT *
FROM    salariat s, department d
WHERE    s.cod_depart = d.cod_depart;
```

ALGEBRA RELAȚIONALĂ

θ-JOIN

- 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:

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

Caz particular de **θ-JOIN**: equi-join (condiția utilizează operatorul “=“)

ALGEBRA RELAȚIONALĂ

Exemplu. Să se afișeze pentru fiecare salariat, codul acestuia și grila sa de salarizare.

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

Exemplu. Să se obțină informații despre contractanți (codul și banca) și obiectivele de investiție asociate acestora (denumire, număr certificat de urbanizare) cu condiția ca obiectivele să nu fie la aceeași adresă ca și contractanții.

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

$$R = \text{PROJECT}(\text{CONTRACTANT}, \text{cod_contractant}, \text{banca});$$
$$S = \text{PROJECT}(\text{OBIECTIV_INVESTITIE}, \text{denumire}, \text{nr_cert_urb});$$
$$\text{Rezultat} = \text{JOIN}(R, S, \text{OBIECTIV_INVESTITIE.adresa} <> \text{CONTRACTANT.adresa}).$$

2. Operatorul **θ -JOIN** în SQL:

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

ALGEBRA RELAȚIONALĂ

SEMI-JOIN

- ▶ Operatorul SEMI-JOIN conservă **atributele unei singure relații participante** la compunere și 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 și S.
- ▶ SEMI-JOIN este un operator derivat, fiind o combinație de proiecție și compunere naturală sau proiecție și 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 attributele relației R .

ALGEBRA RELAȚIONALĂ

Exemplu. Să se obțină informații referitoare la persoanele fizice (nume, carte_identitate) care investesc în obiective cu caracter recreativ.

1. Operatorul SEMI-JOIN în algebra relațională:

$R = \text{SELECT} (\text{OBIECTIV_INVESTITIE}, \text{denumire} = \text{'cabana'} \text{ OR } \text{denumire} = \text{'casa de vacanta'})$

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

Rezultat = **PROJECT** (S, nume, carte_identitate).

2. Operatorul SEMI-JOIN în SQL:

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

ALGEBRA RELAȚIONALĂ

OUTER JOIN

- Operația de **compunere externă** combină tupluri din două relații, tupluri pentru care nu 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 și în cea de-a doua relație**.
 - **Practic, se realizează compunerea a două relații *R* și *S* la care se adaugă tupluri din *R* și *S*, care nu sunt conținute în compunere, completate cu valori *null* pentru attributele care lipsesc.**
- Compunerea externă poate fi: LEFT, RIGHT, FULL. De exemplu, R LEFT OUTER JOIN S 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.

ALGEBRA RELAȚIONALĂ

Exemplu. Să se obțină informații referitoare la persoanele fizice care sunt investitori și la obiectivele lor de investiție industriale. Rezultatul va include și investitorii care nu au investit în obiective industriale.

$R = \text{SELECT} (\text{OBIECTIV_INVESTITIE}, \text{denumire} = \text{'industrial'})$

Rezultat = **LEFT OUTER JOIN** (PERS_FIZICA, R).

Exemplu. Să se obțină informații referitoare la persoanele fizice care sunt investitori și la obiectivele lor de investiție industriale. Rezultatul va include obiectivele industriale care nu sunt construite de persoane fizice.

$R = \text{SELECT} (\text{OBIECTIV_INVESTITIE}, \text{denumire} = \text{'industrial'})$

Rezultat = **RIGHT OUTER JOIN** (PERS_FIZICA, R).

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

$R = \text{SELECT} (\text{OBIECTIV_INVESTITIE}, \text{denumire} = \text{'industrial'})$

Rezultat = **FULL OUTER JOIN** (PERS_FIZICA, R).

- ▶ Cum implementați în SQL?
- Operatorii algebrei relaționale pot fi reprezentați grafic cu ajutorul unor simboluri speciale. → curs!

TEMĂ



Assignment Teams

BAZE DE DATE

CURS 7

Evaluarea și optimizarea interogărilor

Evaluarea și optimizarea interogărilor

Procesarea interogărilor

- 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 apariției lor 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 aceeași relație.

Procesarea interogărilor

Procesarea interogărilor

Exemple referitoare la moduri echivalente de exprimare a unei cereri.

1. Informații despre salariații care nu contribuie la machetarea niciunei publicații, dar au retribuția mai mare decât o valoare dată.
2. Codul și numele subantreprenorilor care au realizat lucrări specialize la obiective case de vacanță sau cabane.
3. Codurile și telefoanele investitorilor, valoarea investiției, durata de execuție a lucrărilor pentru pentru care valoarea investiției este între două limite specificate.
4. Perioada de desfășurare și prețul ofertelor care au început după 1 ianuarie 2019 și sunt:
 - ▶ sejururi la munte;
 - ▶ excursii în care autocarele sunt conduse de șoferi angajați după 1 mai 2007 și supravegheate de ghizi ce cunosc limba engleză care au făcut specializare în Suedia.

Procesarea interogărilor

- În majoritatea sistemelor de gestiune, și în special în cele relationale, interfața cu utilizatorul este de tip **neprocedural**.
 - Utilizatorul definește datele pe care dorește să le vizualizeze fără a da algoritmii de acces (**CE**, nu **CUM**).
 - 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

Evaluarea unei interogări se efectuează în trei etape:

- ❑ **Analiza cererii** – presupune **studierea sintactică și semantică** a cererii pentru a verifica corectitudinea sa și a simplifica criteriul de căutare.
- ❑ **Ordonanțarea** – presupune **descompunerea** cererii **într-o mulțime de operații elementare** și determinarea unei **ordini optimale** a acestor operații. Operațiile sunt, în general, cele ale algebrei relaționale. La sfârșitul etapei se obține un **plan de execuție al cererii**.
- ❑ **Execuția** – presupune **efectuarea** (paralelă și/sau secvențială) operațiilor elementare furnizate de planul de execuție pentru a obține **rezultatul cererii**.

Procesarea interogărilor

- ▶ Presupunem că utilizatorul transmite sistemului de gestiune o **cerere** exprimată prin **comenzi SQL**.
 - ▶ Pentru a răspunde cererii, SGBD-ul trebuie să înțeleagă cererea utilizatorului.
 - ▶ Cererea trebuie să fie:
 - ▶ **corectă sintactic**,
 - ▶ datele trebuie să fie **disponibile** utilizatorului și
 - ▶ trebuie localizate analizând diferite **drumuri de acces** la ele.

Procesarea interogărilor

- ▶ Aceste funcții sunt realizate de SGBD cu ajutorul a **două module funcționale** care comunică permanent:
 - ▶ **analizorul cererilor**, care asigură:
 - ▶ verificarea sintactică și semantică a cererii,
 - ▶ localizarea datelor implicate în cerere (găsirea adresei blocurilor ce conțin datele),
 - ▶ furnizarea planului de execuție.
 - ▶ **administratorul datelor** (executorul), care execută efectiv cererea (primește planurile de execuție furnizate de modulul de optimizare și le execută).
 - ▶ Execuția presupune căutarea blocurilor ce conțin datele și transferul blocurilor în memoria *cache*.

Procesarea interogărilor

Ideea generală:

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

- ▶ Un plan de execuție implică:
 - ▶ **o secvență de pași pentru evaluarea cererii** - în mod obișnuit, fiecare pas din planul de execuție corespunde unei operații relationale
 - ▶ **metoda care va fi folosită pentru evaluarea operației** - de obicei, pentru o operație relațională dată, există mai multe metode ce pot fi folosite pentru evaluarea acesteia.

Procesarea interogărilor

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

Ordinea de execuție a operațiilor

- ▶ O interogare constă dintr-un număr de operații.
 - ▶ Ordinea în care se efectuează operațiile are un rol important în evaluarea costului necesar realizării interogării.
- ▶ Există **două modalități** de abordare pentru a determina ordinea de execuție a operațiilor:
 - ▶ algebric
 - ▶ bazat pe estimarea costului.
- ▶ Ambele folosesc o mulțime de **reguli** care permit **transformarea unui plan de execuție** (reprezentat ca o expresie scrisă în termenii algebrei relaționale) în altul, echivalent.

Ordinea de execuție a operațiilor

- ▶ **Optimizarea cererilor bazată pe algebra relatională** se realizează în două etape:
 - ▶ se exprimă cererile sub forma unor **expresii algebrice** relaționale;
 - ▶ se aplică acestor expresii **transformări algebrice** care conduc la **expresii echivalente**, dar care vor fi **executate mai eficient**.
- ▶ Procesul de transformare a cererilor se realizează conform unei **strategii de optimizare** care poate să fie:
 - ▶ independentă de modul de memorare a datelor (strategie generală) sau
 - ▶ dependentă de modul de memorare (strategie specifică unui anumit SGBD).

Proprietățile operatorilor algebrei relaționale

Proprietatea 1. Comutativitatea operațiilor de *join* și produs cartezian:

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

$$R1 \times R2 = R2 \times R1$$

Proprietatea 2. Asociativitatea operațiilor de *join* și produs cartezian:

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

$$(R1 \times R2) \times R3 = R1 \times (R2 \times R3)$$

Proprietatea 3. Compunerea proiecțiilor:

$$\Pi_{A_1, \dots, A_m} (\Pi_{B_1, \dots, B_n} (R)) = \Pi_{A_1, \dots, A_m} (R),$$

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

Proprietatea 4. Compunerea selecțiilor:

$$\sigma_{cond_1} (\sigma_{cond_2} (R)) = \sigma_{cond_1 \wedge cond_2} (R) = \sigma_{cond_2} (\sigma_{cond_1} (R)),$$

unde am notat prin *cond* condiția după care se face selecția.

Proprietățile operatorilor algebrei relaționale

Proprietatea 5. Comutarea selecției cu proiecția:

$$\Pi_{A_1, \dots, A_m} (\sigma_{cond} (R)) = \sigma_{cond} (\Pi_{A_1, \dots, A_m} (R)),$$

unde condiția după care se face selecția implică numai attributele A_1, \dots, A_m .

Dacă condiția implică și attributele B_1, \dots, B_n , care nu aparțin mulțimii $\{A_1, \dots, A_m\}$, atunci:

$$\Pi_{A_1, \dots, A_m} (\sigma_{cond} (R)) = \Pi_{A_1, \dots, A_m} (\sigma_{cond} (\Pi_{A_1, \dots, A_m, B_1, \dots, B_n} (R)))$$

Proprietățile operatorilor algebrei relaționale

Proprietatea 6. Comutarea selecției cu produsul cartezian:

Dacă toate atributele menționate în condiția după care se face selecția sunt atributе ale relației $R1$, atunci:

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

Dacă condiția este de forma $cond1 \wedge cond2$ și dacă $cond1$ implică numai atributе din $R1$, iar $cond2$ implică numai atributе din $R2$, atunci

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

Dacă $cond1$ implică numai atributе din $R1$, iar $cond2$ implică atributе atât din $R1$ cât și din $R2$, atunci:

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

Proprietățile operatorilor algebrei relaționale

Proprietatea 7. Comutarea selecției cu reuniunea:

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

Proprietatea 8. Comutarea selecției cu diferența:

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

Proprietatea 9. Comutarea proiecției cu produsul cartezian:

Dacă A_1, \dots, A_m este o listă de atribute ce apar în schemele relaționale $R1$ și $R2$ și dacă lista este formată din atribute aparținând lui $R1$ (notate prin B_1, \dots, B_n) și din atribute aparținând lui $R2$ (notate prin C_1, \dots, C_k) atunci:

$$\Pi_{A_1, \dots, A_m}(R1 \times R2) = \Pi_{B_1, \dots, B_n}(R1) \times \Pi_{C_1, \dots, C_k}(R2)$$

Proprietatea 10. Comutarea proiecției cu reuniunea:

$$\Pi_{A_1, \dots, A_m}(R1 \cup R2) = \Pi_{A_1, \dots, A_m}(R1) \cup \Pi_{A_1, \dots, A_m}(R2)$$

Proprietățile operatorilor algebrei relaționale

Proprietatea 11. Compunerea proiecției cu operația *join*:

Dacă A_1, \dots, A_m este o listă de atribute ce apar în schemele relaționale $R1$ și $R2$ și dacă lista este formată din atribute aparținând lui $R1$ (notate prin B_1, \dots, B_n) și din atribute aparținând lui $R2$ (notate prin C_1, \dots, C_k) atunci:

$$\Pi_{A_1, \dots, A_m} (\text{JOIN}(R1, R2, D)) = \Pi_{A_1, \dots, A_m} (\text{JOIN}(\Pi_{D, B_1, \dots, B_n}(R1), \Pi_{D, C_1, \dots, C_k}(R2), D)),$$

unde am notat prin $\text{JOIN}(R1, R2, D)$ operația de compunere naturală între $R1$ și $R2$ după atributul comun D .

Proprietatea 12. Compunerea selecției cu operația *join*:

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

unde A reprezintă atributele care apar în condiția după care se face selecția.

Proprietățile operatorilor algebrei relaționale

Reguli de optimizare frecvent folosite:

- ▶ **Regula de optimizare 1.** **Selecțiile se execută cât mai devreme posibil.** Motivația acestei reguli este că selecțiile reduc substanțial dimensiunea relațiilor. Regula de transformare 4 poate fi folosită pentru a separa două sau mai multe selecții în selecții individuale care pot fi distribuite *join*-ului sau produsului cartezian folosind comutarea selecției cu *join*-ul.
- ▶ **Regula de optimizare 2.** **Produsele carteziene se înlocuiesc cu *join*-uri, ori de câte ori este posibil.** Un produs cartezian între două relații este de obicei mult mai scump (ca și cost) decât un *join* între cele două relații, deoarece primul generează concatenarea tuplurilor în mod exhaustiv și poate genera un rezultat foarte mare. Această transformare se poate realiza folosind legătura dintre produs cartezian, *join* și selecție.

Proprietățile operatorilor algebrei relaționale

- ▶ **Regula de optimizare 3.** **Dacă sunt mai multe *join*-uri atunci cel care se execută primul este cel mai restrictiv.**
 - ▶ Un *join* este mai restrictiv decât altul dacă produce o relație mai mică.
 - ▶ Se poate determina care *join* este mai restrictiv pe baza selectivității sau cu ajutorul informațiilor statistice.
 - ▶ Algebraic, acest lucru se poate realiza folosind regula de transformare 2.
- ▶ **Regula de optimizare 4. Proiecțiile se execută la început pentru a îndepărta atributele nefolositoare.**
 - ▶ Dacă un atribut al unei relații nu este folosit în operațiile ulterioare atunci trebuie îndepărtat. În felul acesta se va folosi o relație mai mică în operațiile ulterioare.
 - ▶ Aceasta se poate realiza folosind comutarea proiecției cu *join*-ul.

BAZE DE DATE

CURS 8

Regulile lui Codd. Normalizare (partea 1).

Regulile lui Codd

Caracteristici ale modelului relațional:

- ▶ nu există tupluri identice;
- ▶ ordinea liniilor și a coloanelor este arbitrară;
- ▶ articolele unui domeniu sunt omogene;
- ▶ fiecare coloană definește un domeniu distinct și nu se poate repeta în cadrul aceleiași relații;
- ▶ toate valorile unui domeniu corespunzătoare tuturor cazurilor nu mai pot fi descompuse în alte valori (sunt atomice).

Regulile lui Codd

Avantajele modelului relațional:

- ▶ fundamentare matematică riguroasă;
- ▶ independență fizică a datelor;
- ▶ posibilitatea filtrărilor;
- ▶ existența unor structuri de date simple;
- ▶ realizarea unei redundanțe minime;
- ▶ suplețe în comunicarea cu utilizatorul neinformatician.

Regulile lui Codd

Avantajele modelului relațional:

- ▶ fundamentare matematică riguroasă;
- ▶ independență fizică a datelor;
- ▶ posibilitatea filtrărilor;
- ▶ existența unor structuri de date simple;
- ▶ realizarea unei redundanțe minime;
- ▶ suplețe în comunicarea cu utilizatorul neinformatician.

Regulile lui Codd

Limite ale modelului relațional:

- ▶ rămâne totuși redundanță,
- ▶ ocupă spațiu,
- ▶ apar fenomene de inconsistență,
- ▶ nu există mecanisme pentru tratarea optimă a cererilor recursive,
- ▶ nu lucrează cu obiecte complexe,
- ▶ nu există mijloace perfecționate pentru exprimarea constrângerilor de integritate,
- ▶ nu realizează gestiunea totală a datelor distribuite,
- ▶ nu realizează gestiunea cunoștințelor.

Regulile lui Codd

- ▶ În anul 1985, E.F. Codd a publicat un set de 13 reguli în raport cu care un sistem de gestiune a bazelor de date poate fi apreciat ca relațional.
 - ▶ Niciun sistem de gestiune a bazelor de date nu respectă absolut toate regulile definite de Codd, dar acest lucru nu împiedică etichetarea acestor sisteme drept relaționale.
- ▶ Nu trebuie apreciat un SGBD ca fiind relațional sau nu, ci **măsura în care acesta este relațional**, deci numărul regulilor lui Codd pe care le respectă.

Regulile lui Codd

- ▶ **Regula 1 – regula gestionării datelor.** *Un SGBD relational trebuie să fie capabil să gestioneze o bază de date numai prin posibilitățile sale relationale.*
- ▶ **Regula 2 – regula reprezentării informației.** *Într-o bază de date relatională, informația este reprezentată la nivel logic sub forma unor **tabele ce poartă numele de relații**.*
- ▶ **Regula 3 – regula accesului garantat la date.** *Fiecare valoare dintr-o bază de date relatională trebuie să poată fi accesată în mod logic printr-o **combinație formată din numele relației, valoarea cheii primare și numele atributului**.*
- ▶ **Regula 4 – regula reprezentării informației necunoscute.** *Un sistem relational trebuie să permită utilizatorului definirea unui tip de date numit „null” pentru reprezentarea unei informații necunoscute la momentul respectiv.*

Regulile lui Codd

- ▶ **Regula 5 – regula dicționarelor de date.** Asupra descrierii bazelor de date (*informații relative la relații, vizualizări, indecși etc.*) trebuie să se poată aplica *aceleași operații ca și asupra datelor din baza de date*.
- ▶ **Regula 6 – regula limbajului de interogare.** Trebuie să existe cel puțin un *limbaj pentru prelucrarea bazei de date*.
- ▶ **Regula 7 – regula de actualizare a vizualizării.** Un SGBD trebuie să poată determina dacă o vizualizare poate fi actualizată și să stocheze rezultatul interogării într-un dicționar de tipul unui catalog de sistem.
- ▶ **Regula 8 – regula limbajului de nivel înalt.** Regulile de prelucrare asupra unei relații luată ca întreg sunt valabile atât pentru operațiile de regăsire a datelor, cât și asupra operațiilor de inserare, actualizare și stergere a datelor.

Regulile lui Codd

- ▶ **Regula 9 – regula independenței fizice a datelor:** *Programele de aplicație și activitățile utilizatorilor nu depind de modul de depunere a datelor sau de modul de acces la date.*
- ▶ **Regula 10 – regula independenței logice a datelor.** *Programele de aplicație trebuie să fie transparente la modificările de orice tip efectuate asupra datelor.*
- ▶ **Regula 11 – regula independenței datelor din punct de vedere al integrității.** *Regulile de integritate trebuie să fie definite într-un sublimbaj relațional, nu în programul de aplicație.*
- ▶ **Regula 12 – regula independenței datelor din punct de vedere al distribuirii.** *Distribuirea datelor pe mai multe calculatoare dintr-o rețea de comunicații de date nu trebuie să afecteze programele de aplicație.*
- ▶ **Regula 13 – regula versiunii procedurale a unui SGBD.** *Orice componentă procedurală a unui SGBD trebuie să respecte aceleași restricții de integritate ca și componenta relațională.*

Regulile lui Codd

- ▶ Deoarece regulile lui Codd sunt prea severe pentru a fi respectate de un SGBD operațional, s-au formulat **criterii minimale de definire a unui sistem de gestiune relațional**.
- ▶ Un SGBD este **minimal relațional** dacă:
 - ▶ toate datele din cadrul bazei sunt reprezentate prin **valori în tabele**;
 - ▶ nu există pointeri observabili de către utilizator;
 - ▶ sistemul suportă operatorii relaționali de **proiecție, selecție și compunere naturală**, fără limitări impuse din considerente interne.
- ▶ Un SGBD este **complet relațional** dacă este minimal relațional și satisfacă încă plus condițiile:
 - ▶ sistemul suportă **restricțiile de integritate de bază** (unicitatea cheii primare, constrângerile referențiale, integritatea entității).
 - ▶ sistemul suportă **toate operațiile de bază ale algebrei relaționale**.

NORMALIZAREA RELAȚIILOR

- ▶ În procesul modelării unei baze de date relaționale, o etapă importantă o reprezintă **normalizarea relațiilor conceptuale** (Codd), adică obținerea de relații „moleculare” fără a pierde nimic din informație pentru a elimina:
 - ▶ redundanță;
 - ▶ anomaliiile reactualizării informațiilor.

NORMALIZAREA RELAȚIILOR

- ▶ Tehnica normalizării permite:
 - ▶ obținerea unei **scheme conceptuale rafinate** printr-un proces de **ameliorare progresivă** a unei scheme conceptuale inițiale a bazei de date relaționale.
- ▶ După fiecare etapă de ameliorare, relațiile bazei de date ating un anumit **grad de perfecțiune**, deci se află într-o anumită **formă normală**.
 - ▶ Trecerea unei relații dintr-o formă normală în alta, presupune **eliminarea unui anumit tip de dependențe** nedorite, care sunt transformate în dependențe admisibile, adică dependențe care nu provoacă anomalii.

NORMALIZAREA RELAȚIILOR

- ▶ Procesul de ameliorare a schemei conceptuale **trebuie**:
 - ▶ să garanteze **conservarea datelor**, adică în schema conceptuală finală trebuie să figureze toate datele din cadrul schemei inițiale;
 - ▶ să garanteze **conservarea dependențelor** dintre date, adică în schema finală fiecare dependență trebuie să aibă determinantul și determinatul în schema aceleiași relații;
 - ▶ să reprezinte o **descompunere minimală** a relațiilor inițiale, adică niciuna dintre relațiile care compun schema finală nu trebuie să fie conținută într-o altă relație din această schemă.

NORMALIZAREA RELAȚIILOR

Există două metode pentru a modela **baze de date relaționale fără anomalii sau pierderi de informație**.

- ▶ **Schema descompunerii** pleacă de la o **schemă relațională universală** ce conține toate atributele BD.
 - ▶ Schema se descompune prin **proiecții succesive** în subrelații.
 - ▶ Descompunerea se oprește când continuarea ei ar duce la pierderi de informație.
 - ▶ Algoritmii de descompunere se bazează, în general, pe descrierea formală a dependenței dintre atrbute.
- ▶ **Schema sintezei** pleacă de la o mulțime de atrbute independente.
 - ▶ Utilizând proprietăți de semantică și legături între atrbute se pot compune noi relații, astfel încât, acestea să nu sufere de anumite anomalii.
 - ▶ Algoritmii se bazează, în general, pe teoria grafurilor pentru a reprezenta legăturile între atrbute.

Dependențe funcționale

- ▶ O **relație universală** este o relație ce grupează toate attributele care modelează sistemul real cercetat.
- ▶ Fie E , mulțimea dependențelor considerate de proiectantul bazei pentru o schemă relațională sau pentru o relație universală.
 - ▶ Plecând de la o mulțime de proprietăți formale ale dependențelor, proprietăți considerate drept reguli de deducție (**axiome**), poate fi obținută **mulțimea maximală de dependențe asociate lui E** . Această mulțime definește **închiderea** lui E .

Dependențe funcționale

- ▶ Fie E mulțimea dependențelor unei relații și $p_1, p_2, \dots, p_r, r \geq 1$, proprietăți formale ale acestor dependențe.
 - ▶ Dacă există o mulțime E' , astfel încât orice dependență a mulțimii E este derivabilă din E' prin aplicarea proprietăților p_1, p_2, \dots, p_r , atunci mulțimea E' definește **acoperirea** lui E pentru proprietățile p_1, p_2, \dots, p_r .
- ▶ E' este **o acoperire minimală** pentru E , dacă nu există nici o submulțime proprie, nevidă a lui E' care să fie o acoperire pentru E . Evident, E și E' au închideri identice, deci dispun de același potențial informațional!

Dependențe funcționale

- ▶ Fie $R(A_1, A_2, \dots, A_n)$ o schemă relațională și fie X, Y submulțimi de attribute ale lui R .
 - ▶ X determină funcțional Y sau Y depinde funcțional (FD) de X , dacă pentru orice relație r (valoare curentă a lui R) nu există două tupluri care să aibă aceleași valori pentru attributele lui X și să aibă valori diferite pentru cel puțin un atribut din Y . Cu alte cuvinte, o valoare a lui X , determină unic o valoare a lui Y .
- ▶ Notație: $X \rightarrow Y$. X este numit **determinant**, iar Y este numit **determinat** (sau dependent).
 - ▶ Dependența funcțională $X \rightarrow Y$ este trivială dacă $Y \subseteq X$.

Dependențe funcționale

- ▶ Comparând toate submultimile de atrbute ale unei relații și determinând legăturile dintre ele, se pot obține toate dependențele funcționale pe care o relație le satisface.
 - ▶ Această abordare **nu** este eficientă, consumând mult timp.
- ▶ Există posibilitatea ca, știind **anumite dependențe** funcționale și utilizând **reguli de deducție**, să fie obținute **toate** dependențele funcționale.

Dependențe funcționale

Fie X, Y, Z, W multimi de atribute ale unei scheme relaționale R și fie următoarele axiome:

- ▶ **Ax1 – reflexivitate.** $X \rightarrow X$. Mai general, dacă $Y \subseteq X$, atunci $X \rightarrow Y$.
- ▶ **Ax2 – creșterea determinantului.** Pot fi considerate următoarele formulări echivalente pentru această axiomă.
 - ▶ Dacă $X \rightarrow Y$ și $X \subseteq Z$, atunci $Z \rightarrow Y$.
 - ▶ Dacă $X \rightarrow Y$ și $W \subseteq Z$, atunci $X \cup Z \rightarrow Y \cup W$.
 - ▶ Dacă $X \rightarrow Y$ atunci $X \cup Z \rightarrow Y \cup Z$.
 - ▶ Dacă $X \rightarrow Y$ atunci $X \cup Z \rightarrow Y$.
- ▶ **Ax3 – tranzitivitate.** Dacă $X \rightarrow Y$ și $Y \rightarrow Z$, atunci $X \rightarrow Z$.

Dependențe funcționale

- ▶ O mulțime de axiome este **completă** dacă și numai dacă plecând de la o mulțime de dependențe E se pot obține toate dependențele închiderii lui E , utilizând axiomele mulțimii.
- ▶ O mulțime de axiome este **închisă** dacă și numai dacă plecând de la o mulțime de dependențe E , nu poate fi dedusă cu ajutorul axiomelor o dependență care nu aparține închiderii lui E . (nu obțin altele!)
- ▶ **Ullman** a demonstrat că axiomele Ax1 – Ax3, numite **axiomele lui Armstrong**, reprezintă o mulțime încisă și completă de axiome. Consecința acestui rezultat este că **închiderea lui E reprezintă mulțimea dependențelor deduse din E , prin aplicarea axiomelor lui Armstrong!!!**

Dependențe funcționale

- ▶ Nu toate dependențele funcționale sunt folosite pentru modelarea relațională.
- ▶ O dependență funcțională $X \rightarrow Y$ se numește **dependență funcțională totală** (FT), dacă și numai dacă nu există nicio submulțime proprie $X' \subset X$, astfel încât $X' \rightarrow Y$.
 - ▶ Dacă există o submulțime proprie $X' \subset X$, astfel încât $X' \rightarrow Y$, atunci dependența funcțională $X \rightarrow Y$ este **parțială**. În axioma Ax2, dependența $Z \rightarrow Y$ este o dependență funcțională parțială.

Dependențe funcționale

- ▶ În cazul dependenței funcționale totale, axiomele lui Armstrong se reduc la o axiomă unică și anume **pseudo-tranzitivitatea**:
 - ▶ dacă $X \rightarrow Y$ și $W \cup Y \rightarrow Z$, atunci $W \cup X \rightarrow Z$.
- ▶ Această axiomă este o regulă de deducție completă pentru total dependențe:
 - ▶ pseudo-tranzitivitatea implică tranzitivitatea ($W = \emptyset$);
 - ▶ reflexivitatea nu poate fi utilizată pentru a obține dependențe totale;
 - ▶ reflexivitatea și pseudo-tranzitivitatea implică creșterea.

Dependențe funcționale

- ▶ Dacă F este o mulțime de dependențe funcționale totale, atunci **închiderea pseudo-tranzitivă** F^+ a acestei mulțimi este reuniunea mulțimilor dependențelor funcționale totale care pot fi obținute din F folosind **axioma de pseudo-tranzitivitate**.
- ▶ Două mulțimi de dependențe funcționale totale sunt **echivalente** dacă au **închideri pseudo-tranzitive identice**.
 - ▶ Pentru a modela scheme relaționale se consideră **mulțimi minimale de dependențe funcționale totale**, capabile să genereze toate închiderile pseudo-tranzitive. Aceste mulțimi definesc **acoperiri minimale**.

Dependențe funcționale

- ▶ O mulțime de dependențe funcționale totale F^* asociată unei mulțimi de atribute A definește o **acoperire minimală** dacă satisface următoarele proprietăți:
 - ▶ nici o dependență funcțională din F^* nu este redundantă;
 - ▶ toate dependențele funcționale totale între submulțimi ale lui A sunt în închiderea pseudo-tranzitivă a lui F^* .
- ▶ Orice mulțime de dependențe totale are cel puțin o acoperire minimală. Alegerea acoperirii minime este punctul de start în modelarea schemelor relationale.

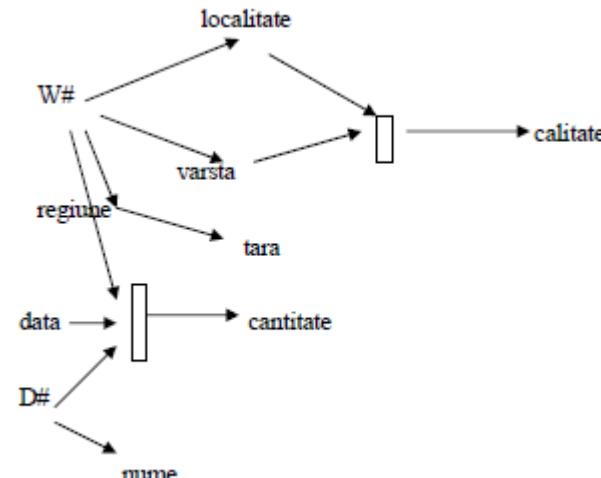
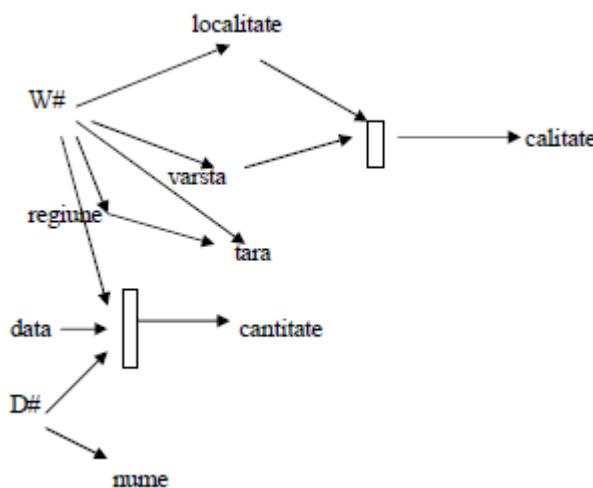
Dependențe funcționale

- ▶ Dependențele funcționale între atributele bazei pot fi reprezentate grafic.
 - ▶ Fie $A = \{A_1, A_2, \dots, A_n\}$ o mulțime de attribute și fie o mulțime de dependențe funcționale $\{X_i \rightarrow A_j\}$, unde X_i este o submulțime a lui A .
- ▶ **Graful dependențelor funcționale** este un **graf direcționat bipartit**, definit astfel:
 1. pentru fiecare atribut A_j există un singur **nod** având eticheta A_j ;
 2. pentru fiecare dependență funcțională de forma $A_i \rightarrow A_j$, există un **arc** de la A_i la A_j ;
 3. pentru fiecare dependență funcțională de forma $X_i \rightarrow A_j$, unde mulțimea X_i este definită de $X_i = \{A_{i_1}, \dots, A_{i_p}\}$ cu $p > 1$, există un **nod auxiliar** etichetat prin X_i și **o mulțime de arce** plecând de la A_{i_1}, \dots, A_{i_p} pentru a obține pe X_i și **printr-un arc adițional** de la X_i la A_j . Nodurile X_i se reprezintă prin dreptunghiuri.

Dependențe funcționale

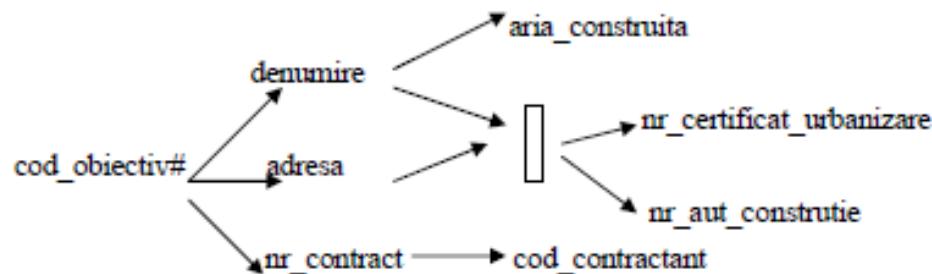
Exemple:

1. Graful dependențelor funcționale pentru schema relațională CONSUMATOR_DE_VIN(W#, localitate, varsta, calitate, regiune, tara, D#, nume, data, cantitate) și acoperirea minimală.



Dependențe funcționale

2. Graful dependențelor funcționale pentru schema relațională OBIECTIV_INVESTITIE.
Dependențele sunt deduse din regulile impuse de beneficiar!



BAZE DE DATE

CURS 9

Normalizare (partea 2)

Necesitatea normalizării

- ▶ Anomaliiile care apar în lucrul cu baza de date se produc din cauza dependențelor care există între datele din cadrul relațiilor bazei de date.
 - ▶ Dependențele sunt plasate greșit în tabele!!!
- ▶ O dependență poate provoca:
 - ▶ **anomalii** la inserare, modificare sau stergere
 - ▶ **redundanță** în date
 - ▶ probleme de **reconexiune**.

Necesitatea normalizării

- ▶ **Normalizarea** are drept **scop**:
 - ▶ suprimarea **redundanței** logice,
 - ▶ evitarea **anomaliiilor** la reactualizare,
 - ▶ rezolvarea **problemei reconexiunii**.
- ▶ Există o **teorie matematică** a normalizării al cărei autor este E.F. Codd.
 - ▶ Soluția: **construirea unor tabele standard** (forme normale).
- ▶ Normalizarea este **procesul reversibil de transformare a unei relații**, în relații de structură mai simplă.
 - ▶ Procesul este reversibil în sensul că **nicio informație nu este pierdută în timpul transformării**.
 - ▶ O relație este într-o formă normală particulară dacă ea satisface o **mulțime specificată de constrângeri**.

Necesitatea normalizării

- ▶ **Relație universală + mulțime de anomalii**
 - ▶ Orice formă normală se obține aplicând o **schemă de descompunere**. Există două tipuri de descompuneri.
 - ▶ **Descompuneri ce conservă dependențele.**
 - ▶ descompunerea relației R în proiecțiile R_1, R_2, \dots, R_k , a.î. dependențele lui R sunt echivalente (au închideri pseudo-tranzitive identice) cu reuniunea dependențelor lui R_1, R_2, \dots, R_k .
 - ▶ **Descompuneri fără pierderi de informație (L -join).**
 - ▶ descompunerea relației R într-o mulțime de proiecții R_1, R_2, \dots, R_j , a.î. pentru orice instanță a lui R este adevărată relația:
- $$R = \text{JOIN}(\Pi_{B1}(R), \Pi_{B2}(R), \dots, \Pi_{Bj}(R))$$
- ▶ Relația inițială = **componerea naturală a relațiilor obținute prin descompunere**.

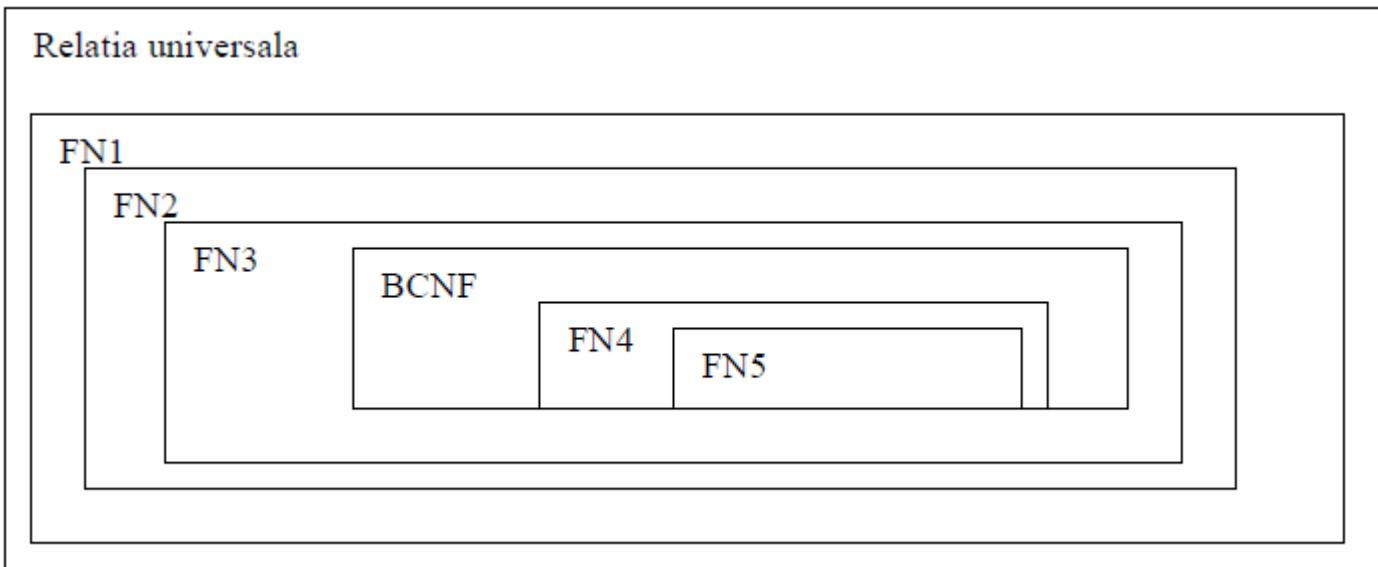
Necesitatea normalizării

- ▶ Formele normale sunt obținute prin descompuneri fără pierderi de informație.
- ▶ O descompunere fără pierdere de informație, utilizată în procesul normalizării, este dată de **regula Casey-Delobel**:
- ▶ Fie $R(A)$ o schemă relațională și fie α , β , γ o partiție a lui A . Presupunem că α determină funcțional pe β . Atunci:

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

- ▶ $\alpha \cup \beta \rightarrow$ mulțimea atributelor care intervin în dependențele funcționale;
- ▶ $\alpha \cup \gamma \rightarrow$ reprezintă reuniunea determinantului cu restul atributelor lui A .

Necesitatea normalizării



Forma normală 1 (FN1)

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

VEHICUL (Non FN1)

cod_persoana#	vehicule
P1	DL, RC, FF
P2	RM, VW
P3	DL

Forma normală 1 (FN1)

Varianta 1:

VEHICUL (FN1)

cod_persoana#	vehicul#
P1	DL
P1	RC
P1	FF
P2	RM
P2	VW
P3	DL

Varianta 2:

VEHICUL (FN1)

cod_persoana#	vehicul1	vehicul2	vehicul3
P1	DL	RC	FF
P2	RM	VW	null
P3	DL	null	null

Forma normală 1 (FN1)

Varianta 3:

VEHICUL 1

cod_persoana#	vehicul
P1	DL
P2	RM
P3	DL

VEHICUL 2

cod_persoana#	vehicul
P1	RC
P2	VW

VEHICUL 3

cod_persoana#	vehicul
P1	FF

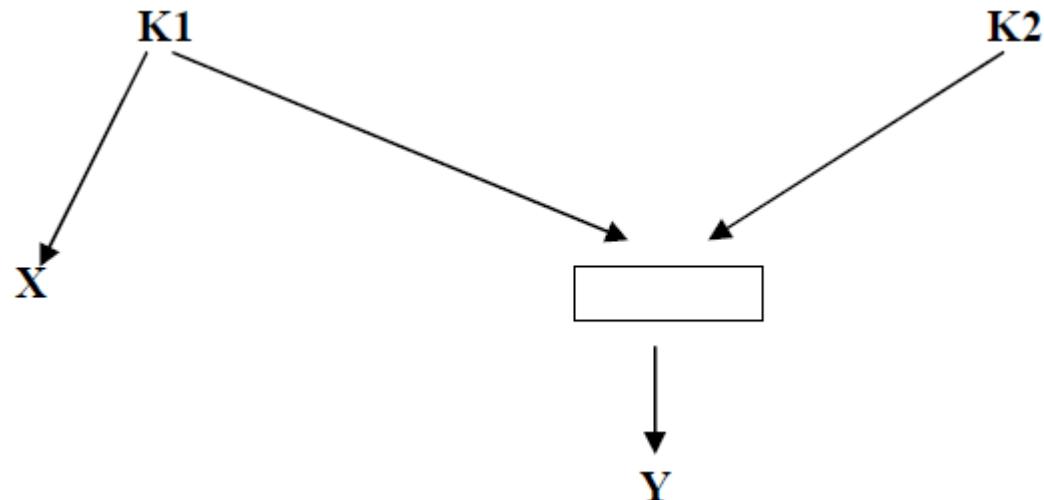
Forma normală 2 (FN2)

- ▶ O relație R este în a doua formă normală dacă și numai dacă:
 - ▶ relația R este în FN1;
 - ▶ fiecare atribut care nu este cheie (nu participă la cheia primară) este dependent de întreaga cheie primară.
- ▶ FN2 interzice manifestarea unor dependențe funcționale parțiale în cadrul relației R !!!
- ▶ Ce consecință imediată se observă?

Forma normală 2 (FN2)

► Aplicarea regulii Casey-Delobel pentru FN2

- Fie relația $R(K1, K2, X, Y)$, unde $K1$ și $K2$ definesc cheia primară, iar X și Y sunt multimi de attribute, astfel încât $K1 \rightarrow X$. Din cauza dependenței funcționale $K1 \rightarrow X$ care arată că R nu este în FN2, se înlocuiește R (fără pierdere de informație) prin două proiecții $R1(K1, K2, Y)$ și $R2(K1, X)$.



Forma normală 2 (FN2)

► Exemplu:

atasat_la

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

De ce nu este în FN2?

Forma normală 2 (FN2)

► Transformarea în FN2:

atasat_2a

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

atasat_2b

Cod_salariat#	Job_cod
S1	Programator
S3	Vanzator
S5	Inginer

Forma normală 2 (FN2)

► Exemplu:

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

LUCRARE(cod_objiectiv#, cod_lucrare#, nume);

SANTIER(nr_santier#, specialitate, sef);

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

- Pentru relația EXECUTA sunt evidente dependențele:

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

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

- Relația EXECUTA este în FN1, dar nu este în FN2. Se aplică regula Casey Delobel:

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

EXECUTA_2(**cod_objiectiv#, cod_lucrare#**, data_inceput, data_sfarsit).

Forma normală 3 (FN3)

- ▶ **Intuitiv**, o relație R este în a treia formă normală dacă și numai dacă:
 - ▶ relația R este în FN2;
 - ▶ fiecare atribut care nu este cheie (nu participă la o cheie) **deinde direct de cheia primară**.
- ▶ Fie R o relație, X o submulțime de attribute ale lui R și A un atribut al relației R . A este **dependent tranzitiv** de X dacă există Y astfel încât $X \rightarrow Y$ și $Y \rightarrow A$ (A nu aparține lui Y și Y nu determină pe X). X nu este dependent funcțional de Y sau A !
 - ▶ De exemplu, dacă $K_1, K_2, K_3 \rightarrow A_1$ și dacă $K_1, K_2, A_1 \rightarrow A_2$, atunci $K_1, K_2, K_3 \rightarrow K_1, K_2, A_1$ și $K_1, K_2, A_1 \rightarrow A_2$. Prin urmare, A_2 este dependent tranzitiv de K_1, K_2, K_3 .

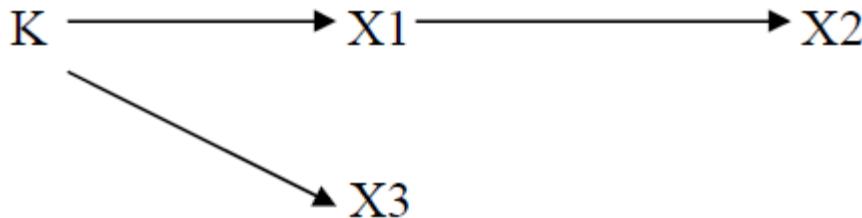
Forma normală 3 (FN3)

- ▶ **Formal**, o relație R este în a treia formă normală dacă și numai dacă:
 - ▶ relația R este în FN2;
 - ▶ fiecare atribut care nu este cheie (nu participă la o cheie) **nu este dependent tranzitiv de nici o cheie a lui R** .
- ▶ O relație este în FN3 dacă și numai dacă fiecare atribut (coloană) care nu este cheie, **depinde de cheie, de întreaga cheie și numai de cheie**.

Forma normală 3 (FN3)

► Aplicarea regulii Casey-Delobel pentru FN3

- Fie relația $R(K, X_1, X_2, X_3)$, unde atributul X_2 depinde tranzitiv de K , iar K este cheia primară a lui R . Presupunem că $K \rightarrow X_1 \rightarrow X_2$.
- Din cauza dependenței funcționale $X_1 \rightarrow X_2$ care arată că R nu este în FN3, se înlocuiește R (fără pierdere de informație) prin două proiecții $R1(K, X_1, X_3)$ și $R2(X_1, X_2)$.



Forma normală 3 (FN3)

► Exemplu:

atasat_2a

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

De ce nu este în FN3?

Forma normală 3 (FN3)

► Transformarea în FN3:

atasat_3a

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

atasat_3b

Functia	Suma
Supervizor	60
Cercetator	25
Auxiliar	10

Forma normală 3 (FN3)

► Exemplu:

În tabelul EXECUTA1(*cod_objiectiv#*, *cod_lucrare#*, *nr_santier#*, *descriere*, *functie*, *conducator*) continuă să existe redundanță în date.

- Atributul *conducator* depinde indirect de cheia primară prin intermediul atributului *functie*. Între attributele relației există dependențele:

$\{cod_obiectiv\#, cod_lucrare\#, nr_santier\} \rightarrow \{descriere\}$,

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

- Se aplică regula Casey-Delobel. Relația se descompune, prin eliminarea dependențelor funcționale tranzitive, în proiecțiile:

EXECUTA11(*cod_objiectiv#*, *cod_lucrare#*, *nr_santier#*, *descriere*, **functie**)

EXECUTA12(**functie**, **conducator**).

Schema de sinteză pentru obținerea lui FN3

- ▶ **Algoritmul de sinteză** construiește o acoperire minimală F a dependentelor funcționale totale.
 - ▶ Se elimină atributele și dependențele funcționale **redundante**.
 - ▶ Mulțimea F este partionată în grupuri F_i , astfel încât în fiecare grup F_i sunt dependențe funcționale care au **același membru stâng** și nu există două grupuri având același membru stâng.
 - ▶ Fiecare grup F_i produce o **schemă FN3**.
- ▶ Algoritmul realizează o descompunere ce conservă dependențele.

Schema de sinteză pentru obținerea lui FN3

- **Algoritm SNF3** (aducerea unei relații în FN3 prin utilizarea unei scheme de sinteză):
 1. Se determină **F o acoperire minimală a lui E** (mulțimea dependențelor funcționale).
 2. Se descompune mulțimea F în **grupuri notate F_i** , astfel încât în cadrul fiecărui grup să existe dependențe funcționale **având aceeași parte stângă**.
 3. Se determină **perechile de chei echivalente (X, Y)** în raport cu F (două mulțimi de attribute X, Y sunt chei echivalente dacă în mulțimea de dependențe E există atât dependența $X \rightarrow Y$, cât și dependența $Y \rightarrow X$).

Schema de sinteză pentru obținerea lui FN3

4. Pentru fiecare pereche de chei echivalente: se identifică grupurile F_i și F_j care conțin dependențele funcționale cu partea stângă X și respectiv Y ; se formează **un nou grup de dependențe** F_{ij} , care va conține dependențele funcționale având **membrul stâng (X, Y)**; se elimină grupurile F_i și F_j , iar locul lor va fi luat de grupul F_{ij} .
5. Se determină o **acoperire minimală a lui F** , care va include toate dependențele $X \rightarrow Y$, unde X și Y sunt chei echivalente (celelalte dependențe sunt redundante).
6. Se construiesc **relații FN3** (câte o relație pentru fiecare grup de dependențe funcționale).

Schema de sinteză pentru obținerea lui FN3

- ▶ Se observă că **algoritmul solicită**:
 - ▶ determinarea unei **acoperiri minime** (algoritmii **EAR** și **EDF**);
 - ▶ determinarea **închiderii (A^+) unei mulțimi de atrbute A** în raport cu mulțimea de dependențe funcționale E (algoritm **AIDF**).
- ▶ Determinarea acoperirii minime presupune eliminarea atrbutelor și dependențelor redundante.
- ▶ Acoperirea minimală **nu este unică** și depinde de ordinea în care sunt eliminate aceste atrbute și dependențe redundante.

Schema de sinteză pentru obținerea lui FN3

- ▶ **Algoritm EAR** (elimină atributele redundante din determinantul dependențelor funcționale)
- ▶ Pentru fiecare dependență funcțională din E și pentru fiecare atribut din partea stângă a unei dependențe funcționale:
 - ▶ **Pas1.** Se elimină atributul considerat.
 - ▶ **Pas2.** Se calculează închiderea părții stângi reduse.
 - ▶ **Pas3.** Dacă închiderea conține toate atributele din determinantul dependenței, atunci atributul eliminat la pasul 1 este redundant și rămâne eliminat. În caz contrar, atributul nu este redundant și se reintroduce în partea stângă a dependenței funcționale.

Schema de sinteză pentru obținerea lui FN3

- ▶ **Algoritm EDF** (elimină dependențele funcționale redundante din E)
- ▶ Pentru fiecare dependență funcțională $X \rightarrow Y$ din E:
 - ▶ **Pas1.** Se elimină dependența din E.
 - ▶ **Pas2.** Se calculează închiderea X^+ , în raport cu mulțimea redusă de dependențe.
 - ▶ **Pas3.** Dacă Y este inclus în X^+ , atunci dependența $X \rightarrow Y$ este redundantă și rămâne eliminată. În caz contrar, se reintroduce în E.

Schema de sinteză pentru obținerea lui FN3

- ▶ **Algoritm AIDF** (determină închiderea lui A)
 - ▶ **Pas1.** Se caută dacă există în E dependențe $X \rightarrow Y$ pentru care determinantul X este o submulțime a lui A, iar determinatul Y nu este inclus în A.
 - ▶ **Pas2.** Pentru fiecare astfel de dependență funcțională se adaugă mulțimii A attributele care constituie determinatul dependenței.
 - ▶ **Pas3.** Dacă nu mai există dependențe funcționale de tipul de la pasul 1, atunci $A^+ = A$.
- ▶ Exemplu!

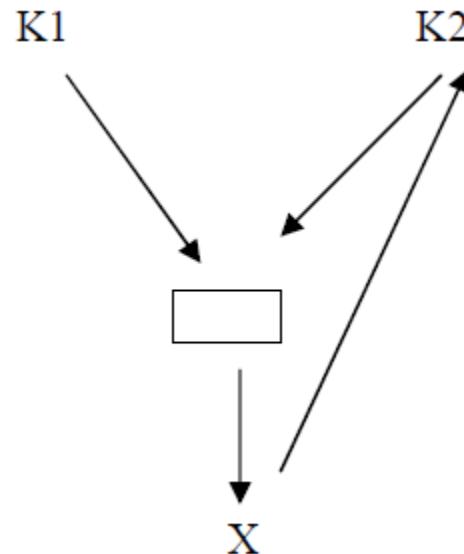
Forma normală Boyce-Codd (BCNF)

- ▶ Determinantul este un atribut sau o mulțime de attribute neredundante, care constituie un identificator unic pentru alt atribut sau altă mulțime de attribute ale unei relații date.
- ▶ **Intuitiv**, o relație R este în forma normală Boyce-Codd dacă și numai dacă **fiecare determinant este o cheie candidat**.
- ▶ **Formal**, o relație R este în forma normală Boyce-Codd dacă și numai dacă pentru orice dependență funcțională totală $X \rightarrow A$, X este o cheie (candidat) a lui R .

Forma normală Boyce-Codd (BCNF)

- Regula Casey Delobel pentru $R(K1\#, K2\#, X)$ presupunând că există dependență: $X \rightarrow K2$.

→ $R1(K1\#, X)$ și $R2(X\#, K2)$



Forma normală Boyce-Codd (BCNF)

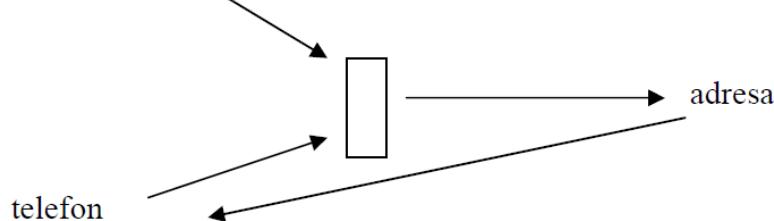
- ▶ Pentru ca o relație să fie adusă în BCNF **nu trebuie în mod obligatoriu să fie în FN3**.
- ▶ Se pot aduce în BCNF și relații aflate în FN1 sau FN2.
 - ▶ Acest lucru este posibil încât dependențele funcționale parțiale și cele tranzitive sunt tot dependențe noncheie, adică dependențe ai căror determinanți nu sunt chei candidat.

Forma normală Boyce-Codd (BCNF)

► Exemplu:

ADRESA(cod_persoana#, telefon#, adresa)

cod_persoana



► În dependența *adresa_ _Telefon* se observă că determinantul nu este o cheie candidat. Relația ADRESA se descompune în:

ADRESA_1(cod_persoana#, adresa);

ADRESA_2(adresa#, telefon).

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

Forma normală Boyce-Codd (BCNF)

- Exemplu:
- Relația INVESTESTE_IN leagă entitățile INVESTITOR și OBIECTIV_INVESTITIE. Ea are schema relațională:

INVESTESTE_IN(cod_contractant#, cod_objiectiv#, nr_contract, cota_parte).

- Între atributele relației există dependențele:

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

$\{nr_contract\} \rightarrow \{cod_obiectiv\}$.

- Se aplică regula Casey-Delobel și se aduce relația în BCNF.

INVESTESTE_IN_1(cod_objiectiv, nr_contract#);

INVESTESTE_IN_2(cod_contractant#, nr_contract, cota_parte).

Forma normală Boyce-Codd (BCNF)

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

1. Dacă relația conține cel mult două attribute, atunci R este în BCNF și algoritmul s-a terminat.
2. Dacă relația conține mai mult de două attribute, se consideră toate perechile (X, Y) de attribute distincte din A .
3. Se determină A_1^+ , închiderea mulțimii $A_1 = A - \{X, Y\}$.
4. Dacă pentru orice pereche (X, Y) , $X \notin A_1^+$ atunci relația R este în BCNF și algoritmul s-a terminat.
5. În caz contrar (pentru cel puțin o pereche (X, Y) , X aparține lui A_1^+), relația R nu este în BCNF.
6. Se reduce progresiv schema relației și se reia algoritmul, exploatând relația redusă. Orice relație obținută prin reducerea lui R și care este în BCNF se consideră ca făcând parte din descompunerea lui R în procesul aducerii sale în BCNF.

Forma normală 4 (FN4)

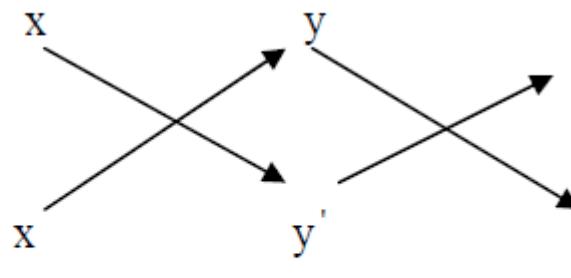
- ▶ FN4 elimină redundanțele datorate relațiilor $m:n$, adică datorate **dependenței multiple**.
- ▶ **Intuitiv**, o relație R este în a patra formă normală dacă și numai dacă relația este în BCNF și nu conține relații $m:n$ independente.

Forma normală 4 (FN4)

- ▶ Fie R o relație definită pe o mulțime de attribute $A = \{A_1, A_2, \dots, A_n\}$ și fie $X, Y, Z \subset A$. Se spune că X **multidetermină** pe Z sau că Z este multidependent de X :
 - ▶ dacă pentru fiecare valoare a lui Z în R există numai o valoare pentru perechea (X, Y) ;
 - ▶ dacă valoarea lui Z depinde numai de valoarea lui X .
- ▶ Acest tip de dependență, numită și multivaloare sau multidependență (MVD) se notează prin $X \rightarrow\rightarrow Z$.
- ▶ **Intuitiv, multidependența reprezintă situația în care valoarea unui atribut (sau a unei mulțimi de attribute) determină o mulțime de valori a altui atribut (sau mulțimi de attribute)!!!**

Forma normală 4 (FN4)

- Multidependența $X \rightarrow\!\!\! \rightarrow Y$ poate fi gândită ca o **regulă de deducție**:
 - dacă tuplurile $\langle x, y, z \rangle$ și $\langle x, y', z' \rangle$ sunt în relație la un moment r , atunci la momentul r sunt în relație și tuplurile $\langle x, y, z' \rangle$ și $\langle x, y', z \rangle$.



x	y	z
x'	y'	z'

Forma normală 4 (FN4)

- ▶ Orice dependentă funcțională este o multidependență.
- ▶ Afirmația inversă nu este adevărată.
- ▶ Dacă $X \rightarrow Y$ (FD), atunci pentru oricare două tupluri $\langle x, y, z \rangle$ și $\langle x, y', z' \rangle$, se obține $y = y'$. Prin urmare în relație apar tuplurile $\langle x, y', z \rangle$ și $\langle x, y, z' \rangle$ și deci $X \rightarrow\rightarrow Y$ (MVD).

Forma normală 4 (FN4)

- ▶ Fie W, V, X, Y și Z submulțimi de attribute ale unei scheme relationale R . Fiind dată o mulțime T de multidependențe există o mulțime completă de axiome (Ax1–Ax8) care permit obținerea tuturor multidependențelor ce se pot deduce din mulțimea T . *-> Vezi curs!*
- ▶ O **multidependență elementară** este o multidependență care are părți stângi și drepte minimale (nu există $X' \subset X$ și $Y' \subset Y$ a.i. $X' \rightarrow\!\!\!\rightarrow Y'$).
- ▶ **Formal**, relația R este în a patra formă normală dacă și numai dacă:
 - ▶ R este în BCNF;
 - ▶ orice dependență multivaloare este o dependență funcțională.

Forma normală 4 (FN4)

- ▶ O relație BCNF este în FN4 dacă pentru orice multidependență elementară de forma $X \rightarrow\!\!\!\rightarrow Y$, X este o supercheie a lui R .
- ▶ **Regula de descompunere în relații FN4.**
 - ▶ Fie $R(X, Y, Z)$ o schemă relațională care nu este în FN4 și fie $X \rightarrow\!\!\!\rightarrow Y$ o multidependență elementară care nu este de forma „CHEIE $\rightarrow\!\!\!\rightarrow$ atribut”.
 - ▶ Această relație este descompusă prin proiecție în două relații:

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

Forma normală 4 (FN4)

- ▶ O relație BCNF este în FN4 dacă pentru orice multidependență elementară de forma $X \rightarrow\!\!\!\rightarrow Y$, X este o supercheie a lui R .
- ▶ **Regula de descompunere în relații FN4.**
 - ▶ Fie $R(X, Y, Z)$ o schemă relațională care nu este în FN4 și fie $X \rightarrow\!\!\!\rightarrow Y$ o multidependență elementară care nu este de forma „CHEIE $\rightarrow\!\!\!\rightarrow$ atribut”.
 - ▶ Această relație este descompusă prin proiecție în două relații:

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

Forma normală 4 (FN4)

► Exemplu:

► Fie relația INVESTITIE(cod_contractant#, denumire#, telefon#) și presupunem că un investitor poate avea mai multe numere de telefon și că poate investi în mai multe obiective.

► Între atributele relației există multidependențele:

$\text{cod_contractant\#} \rightarrow\!\!\! \rightarrow \text{denumire}$;

$\text{cod_contractant\#} \rightarrow\!\!\! \rightarrow \text{telefon}$.

► Relația INVESTITIE este în BCNF. Pentru a aduce relația în FN4 o vom descompune prin proiecție în două relații:

INVESTITIE_1(cod_contractant#, denumire#),

INVESTITIE_2(cod_contractant#, telefon#).

► $\text{INVESTITIE} = \text{JOIN}(\text{INVESTITIE_1}, \text{INVESTITIE_2})$.

Forma normală 5 (FN5)

- ▶ FN5 își propune eliminarea redundanțelor care apar în relații $m:n$ dependente.
 - ▶ În general, aceste relații nu pot fi descompuse.
 - ▶ S-a arătat că o relație de tip 3 este diferită de trei relații de tip 2. Există totuși o excepție, și anume, dacă relația este ciclică
- ▶ **Intuitiv**, o relație R este în forma normală 5 dacă și numai dacă:
 - ▶ relația este în FN4;
 - ▶ nu conține dependențe ciclice.

Concluzii

1. **FN1 → FN2** elimină redundanțele datorate dependenței netotale a atributelor care nu participă la o cheie, față de cheile lui R . Se suprimă dependențele funcționale care nu sunt totale.
2. **FN2 → FN3** elimină redundanțele datorate dependenței tranzitive. Se suprimă dependențele funcționale tranzitive.
3. **FN3 → BCNF** elimină redundanțele datorate dependenței funcționale. Se suprimă dependențele în care partea stângă nu este o supercheie.
4. **BCNF → FN4** elimină redundanțele datorate multidependenței. Se suprimă toate multidependențele care nu sunt și dependențe funcționale.
5. **FN4 → FN5** elimină redundanțele datorate dependenței ciclice. Se suprimă toate *join*-dependențele care nu sunt implicate de o cheie.
6. **BCNF, FN4 și FN5** corespund la regula că orice determinant este o cheie, dar de fiecare dată dependența cu care se definește determinantul este alta și anume dependența funcțională, multidependența sau *join*-dependența).
7. Descompunerea unei relații FN2 în FN3 conservă datele și dependențele, pe când descompunerea unei relații FN3 în BCNF și, respectiv, a unei relații BCNF în FN4 conservă doar datele.

BAZE DE DATE

CURS 10

Normalizare (partea 3). Denormalizare.

Forma normală 5 (FN5)

- ▶ FN5 își propune eliminarea redundanțelor care apar în relații $m:n$ dependente.
 - ▶ În general, aceste relații nu pot fi descompuse.
 - ▶ S-a arătat că o relație de tip 3 este diferită de trei relații de tip 2. Există totuși o excepție, și anume, dacă relația este ciclică
- ▶ **Intuitiv**, o relație R este în forma normală 5 dacă și numai dacă:
 - ▶ relația este în FN4;
 - ▶ nu conține dependențe ciclice.

Forma normală 5 (FN5)

- ▶ Dependența funcțională și multidependența permit descompunerea prin proiecție, fără pierdere de informație, a unei relații în **două relații**.
- ▶ Regulile de descompunere (FN1 – FN4) nu dau toate descompunerile posibile prin proiecție ale unei relații.
- ▶ Există relații care nu pot fi descompuse în două relații dar pot fi descompuse în **trei, patru sau mai multe relații fără a pierde informații**.
- ▶ Pentru a obține descompuneri L-join în trei sau mai multe relații, s-a introdus conceptul de **join-dependență sau dependență la compunere (JD)**.

Forma normală 5 (FN5)

- ▶ Fie $\{R_1, R_2, \dots, R_p\}$ o mulțime de scheme relationale care nu sunt disjuncte și a căror reuniune este R .
- ▶ R satisfacă **join-dependența** $*\{R_1, R_2, \dots, R_p\}$ dacă la fiecare moment are loc egalitatea:

$$R = \text{JOIN}(\Pi_{\alpha_1}(R), \Pi_{\alpha_2}(R), \dots, \Pi_{\alpha_p}(R))$$

unde α_k reprezintă mulțimea atributelor corespunzătoare lui R_k ($1 \leq k \leq p$).

- ▶ *Join-dependența* $*\{R_1, R_2, \dots, R_p\}$ are loc în R , dacă R_1, R_2, \dots, R_p este o descompunere **L-join** a lui R .
- ▶ Pentru $p = 2$ se regăsește **multidependența**.
- ▶ O *join-dependență* $*\{R_1, R_2, \dots, R_p\}$ în care una dintre R_i este chiar R , definește o **join-dependență trivială**.

Forma normală 5 (FN5)

- ▶ *Join*-dependența **generalizează multidependența**.
- ▶ Într-adevăr, multidependența $X \rightarrow\rightarrow Y$ în relația $R(X, Y, Z)$ (deci și $X \rightarrow\rightarrow Z$), corespunde *join*-dependenței $*\{X \cup Y, X \cup Z\}$.
- ▶ Invers, *join*-dependența $*\{R_1, R_2\}$ corespunde multidependenței $R_1 \cap R_2 \rightarrow\rightarrow R_1 - (R_1 \cap R_2)$.
- ▶ Formal, o relație R este în FN5 dacă și numai dacă orice *join* dependență $*\{R_1, R_2, \dots, R_p\}$ care are loc în R fie este trivială, fie conține o supercheie a lui R (adică, o anumită componentă R_i este o supercheie a lui R).
- ▶ Cu alte cuvinte, o relație R este în FN5 dacă orice *join*-dependență definită pe R este implicată de cheile candidat ale lui R .

Forma normală 5 (FN5)

- ▶ Între multimile de attribute X , Y și Z din cadrul relației R există o *join* dependență dacă există multidependențe între fiecare dintre perechile de multimi (X, Y) , (Y, Z) și (X, Z) .
- ▶ Aducerea în FN5 prin eliminarea join dependențelor!

Forma normală 5 (FN5)

- ▶ Între multimile de attribute X , Y și Z din cadrul relației R există o *join* dependență dacă există multidependențe între fiecare dintre perechile de multimi (X, Y) , (Y, Z) și (X, Z) .
- ▶ Aducerea în FN5 prin eliminarea join dependențelor!
- ▶ Exemple – suport curs.

Concluzii NORMALIZARE

1. **FN1 → FN2** elimină redundanțele datorate dependenței netotale a atributelor care nu participă la o cheie, față de cheile lui R . Se suprimă dependențele funcționale care nu sunt totale.
2. **FN2 → FN3** elimină redundanțele datorate dependenței tranzitive. Se suprimă dependențele funcționale tranzitive.
3. **FN3 → BCNF** elimină redundanțele datorate dependenței funcționale. Se suprimă dependențele în care partea stângă nu este o supercheie.
4. **BCNF → FN4** elimină redundanțele datorate multidependenței. Se suprimă toate multidependențele (non-cheie) care nu sunt și dependențe funcționale.
5. **FN4 → FN5** elimină redundanțele datorate dependenței ciclice. Se suprimă toate *join*-dependențele care nu sunt implicate de o cheie.
6. **BCNF, FN4 și FN5** corespund la regula că orice determinant este o cheie, dar de fiecare dată dependența cu care se definește determinantul este alta și anume dependența funcțională, multidependența sau *join*-dependența).
7. Descompunerea unei relații FN2 în FN3 conservă datele și dependențele, pe când descompunerea unei relații FN3 în BCNF și, respectiv, a unei relații BCNF în FN4 conservă doar datele.

DENORMALIZARE

- ▶ Procedura de normalizare elimină redundanțele prin efectuarea unor proiecții, DAR NU toate redundanțele pot fi eliminate în acest mod.

=>Uneori este necesară **denormalizarea** care presupune:

- ▶ Mărirea redundanței;
- ▶ Reducerea numărului de join-uri care trebuie efectuate => micșorare timp de execuție!

DENORMALIZARE

- ▶ Ideile normalizării sunt utile în proiectarea BD, dar **nu sunt obligatorii!**
- ▶ Dependența și normalizarea sunt de natură semantică (cu alte cuvinte, se referă la ceea ce înseamnă datele).
- ▶ În schimb, algebra relațională și calculul relațional (limbajele SQL) se referă doar la valorile efective ale datelor și în multe cazuri nu necesită mai mult decât FN1.

DENORMALIZARE

A) Obiectivul denormalizării constă în **reducerea numărului de join-uri** care trebuie efectuate pentru rezolvarea unei interogări, prin realizarea unora dintre acestea în avans, ca făcând parte din proiectarea bazei de date.

DENORMALIZARE

B) Conceptul de denormalizare suferă de un număr de probleme binecunoscute.

- ▶ Odată începută denormalizarea, nu este clar unde trebuie să se oprească.
- ▶ Nu se mai lucrează cu relații normalize și astfel pot apărea anomaliiile pe care normalizarea le corectează.
- ▶ Un design fizic poate fi bun pentru anumite aplicații, dar prost pentru altele (denormalizarea la nivelul fișierelor stocate).

DENORMALIZARE

Exemplu:

- ▶ Se poate presupune că fiecare tabel corespunde unui anumit fișier stocat și că fiecare fișier stocat este format dintr-o mulțime contiguă fizic de înregistrări stocate, câte una pentru fiecare tuplu din tabel.
- ▶ Se presupune că informațiile (join-ul) despre creatori, vestimentații și accesorii se reprezintă printr-un tabel și, prin urmare, un fișier stocat. În această structură fizică interogarea « Obținerea informațiilor despre creatorii care oferă vestimentații cu accesorii din margele » se rezolvă cu ușurință.

DENORMALIZARE

- ▶ Interogarea « Obținerea informațiilor despre creatorii din Sibiu » va prezenta performanțe mai reduse în această structură fizică, decât dacă am fi menținut trei tabele de bază pentru cele 3 entități și, prin urmare, 3 fișiere stocate fizic separat.
- ▶ În acest ultim design, toate înregistrările despre creatori vor fi fizic contigüe, în timp ce în primul design ele sunt dispersate fizic într-o zonă largă și, prin urmare, vor necesita **mai multe operații I/O**.

DENORMALIZARE

Când este utilă denormalizarea?

- ▶ Ca o regulă empirică, se poate afirma că, dacă performanțele nu sunt satisfăcătoare și relația are o rată de reactualizare scăzută, dar o rata a interogărilor foarte ridicată, denormalizarea poate constitui o opțiune viabilă.
 - ▶ BD de tip OLTP – preponderent normalizare
 - ▶ BD de tip OLAP – preponderent denormalizare
- ▶ Nu există reguli fixe pentru stabilirea situațiilor în care este indicată denormalizarea relațiilor.
- ▶ În general, denormalizarea înseamnă considerarea dublării atributelor și grupării relațiilor
 - ▶ Dublarea (duplicarea) atributelor sau gruparea relațiilor are ca scop reducerea numărului de join-uri necesare pentru efectuarea unei interogări

Exemplu de denormalizare

- ▶ FILIALA (cod_filiala#, strada, zona, oras, cod_postal, telefon)
- ▶ Relația nu este în FN3 deoarece cod_postal → {zona, oras}
- ▶ Aplicăm FN3 și se obține:
 - ▶ FILIALA1(cod_filiala#, strada, cod-postal, telefon)
 - ▶ COD_P(cod_postal#, zona, oras)
- ▶ Nu este convenabil, deoarece rareori vom accesa adresa filialei, fără informații referitoare la zonă și oraș. Prin urmare, în acest caz putem prefera varianta denormalizată (cea aflată în FN2)

Cazuri de denormalizare

- ▶ Considerarea datelor derivate
- ▶ Combinarea relațiilor de tip 1:1
- ▶ Duplicarea atributelor care nu sunt chei în relații 1:M
- ▶ Tabele de căutare
- ▶ Duplicarea atributelor cheii externe într-o relație 1:M pentru simplificarea join-urilor
- ▶ Duplicarea atributelor în relațiile de tip M:N, pentru reducerea join-urilor

Cazuri de denormalizare

Duplicarea atributelor care nu sunt chei în relații 1:M

```
SELECT p.*, pp.nume  
FROM proprietate_de_inchiriat p, proprietar pp  
WHERE p.cod_proprietar = pp.cod AND cod_filiala = 'S3';
```

- Dacă se va duplica atributul nume în relația *proprietate_de_inchiriat*, interogarea devine:

```
SELECT p.*  
FROM proprietate_de_inchiriat p  
WHERE cod_filiala = 'S3';
```

- Avantaje sau dezavantaje? Depinde de problemele care pot apărea.

Cazuri de denormalizare

Tabele de căutare (de referință)

- ▶ Acestea conțin, de obicei, un cod și o descriere (și / sau denumire)
- ▶ Le întâlnim în forma normalizată a bazei de date
- ▶ De exemplu, se poate defini un tabel de căutare pentru tipul de proprietate și modifica tabelul *proprietate_de_inchiriat* astfel:

TIP_PROPRIETATE(cod_tip#, descriere)

PROPRIETATE_DE_INCHIRIAT(cod#, strada, zona, oras, cod_postal, cod_tip,
nr_camere, chirie, cod_proprietar, cod_filiala, nr_personal)

Cazuri de denormalizare

Tabele de căutare (de referință) - continuare

- ▶ Avantaje:
 - ▶ Dacă este modificată descrierea, atunci se va modifica o singură dată, în tabelul de căutare
 - ▶ Se reduce dimensiunea relației de la capătul M al relației (PROPRIETATE_DE_INCHIRIAT)
- ▶ Varianta de pe slide-ul anterior este în FN3
- ▶ Dacă se consideră că accesarea descrierii tipului de proprietate are loc frecvent odată cu accesarea informațiilor despre proprietate, atunci putem denormaliza astfel:

~~TIP_PROPRIETATE(cod_tip#, descriere)~~

PROPRIETATE_DE_INCHIRIAT(cod#, strada, zona, oras, cod_postal, **descriere_tip**, nr_camere, chirie, cod_proprietar, cod_filiala, nr_personal)

Cazuri de denormalizare

Duplicarea atributelor cheii externe într-o relație de tip 1:M

- Obiectiv: simplificarea join-urilor
- Relațiile pot să nu fie înceinate în diagramă
- **Exemplu:** Să se enumere proprietarii de proprietăți de închiriat dintr-o filială.

```
SELECT pp.nume  
FROM proprietate_de_inchiriat p, proprietar pp  
WHERE p.cod_proprietar = pp.cod AND cod_filiala = 'S3';
```

- Dacă se duplică cheia externă *cod_filiala* în relația PROPRIETAR, adică se introduce o relație directă între FILIALA și PERSONAL, atunci cererea devine:

```
SELECT pp.nume  
FROM proprietar pp  
WHERE cod_filiala = 'S3';
```

Cazuri de denormalizare

Duplicarea atributelor cheii externe într-o relație de tip 1:M (continuare)

- ▶ **Atenție!** Sunt necesare constrângeri suplimentare asupra cheilor externe. De exemplu, dacă un proprietar ar închiria prin mai multe filiale atunci modificările nu mai sunt valabile.
 - ▶ Prezența unui atribut multiplu ar însemna non-FN1.
- ▶ Observație: Singurul motiv pentru care relația PROPRIETATE_DE_INCHIRIAT conține atributul *cod_filiala* constă în faptul că este posibil ca o proprietate să nu aibă alocat un membru de personal, mai ales la început, atunci când este preluată de către agenție.

Cazuri de denormalizare

Duplicarea atributelor în relațiile de tip M:N

- ▶ Presupunem că relația M:N dintre CHIRIAS și PROPRIETATE_DE_INCHIRIAT a fost descompusă prin introducerea relației intermediare VIZITARE.
- ▶ **Exemplu:** Care sunt chiriașii care au vizitat proprietăți, dar mai au de făcut comentarii asupra unora dintre ele? Personalul de la agenție are nevoie de atributul *strada* atunci când discută cu chiriașii.

```
SELECT p.strada, c.*, v.data  
  
FROM    chirias c, vizitare v, proprietate_de_inchiriat p  
  
WHERE  v.cod_proprietate = p.cod AND c.cod = v.cod_chirias AND  
comentarii IS NULL;
```

Cazuri de denormalizare

Duplicarea atributelor în relațiile de tip M:N (continuare)

- Dacă se introduce atributul strada în relația VIZITARE, atunci cererea devine:

```
SELECT v.strada, c.*, v.data  
FROM    chirias c, vizitare v  
WHERE   c.cod = v.cod_chirias AND comentarii IS NULL;
```