



BAZE DE DATE

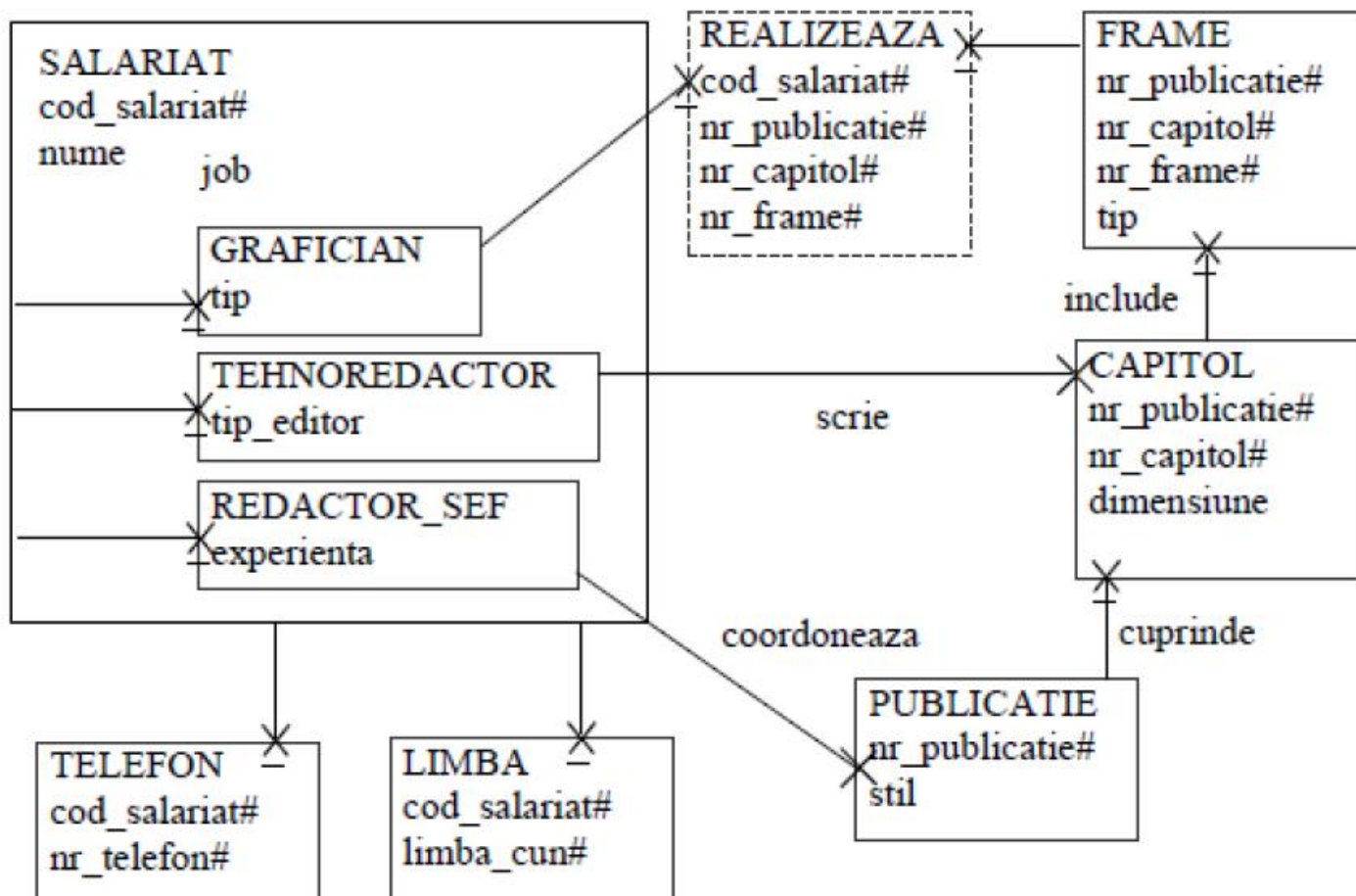
CURS 8

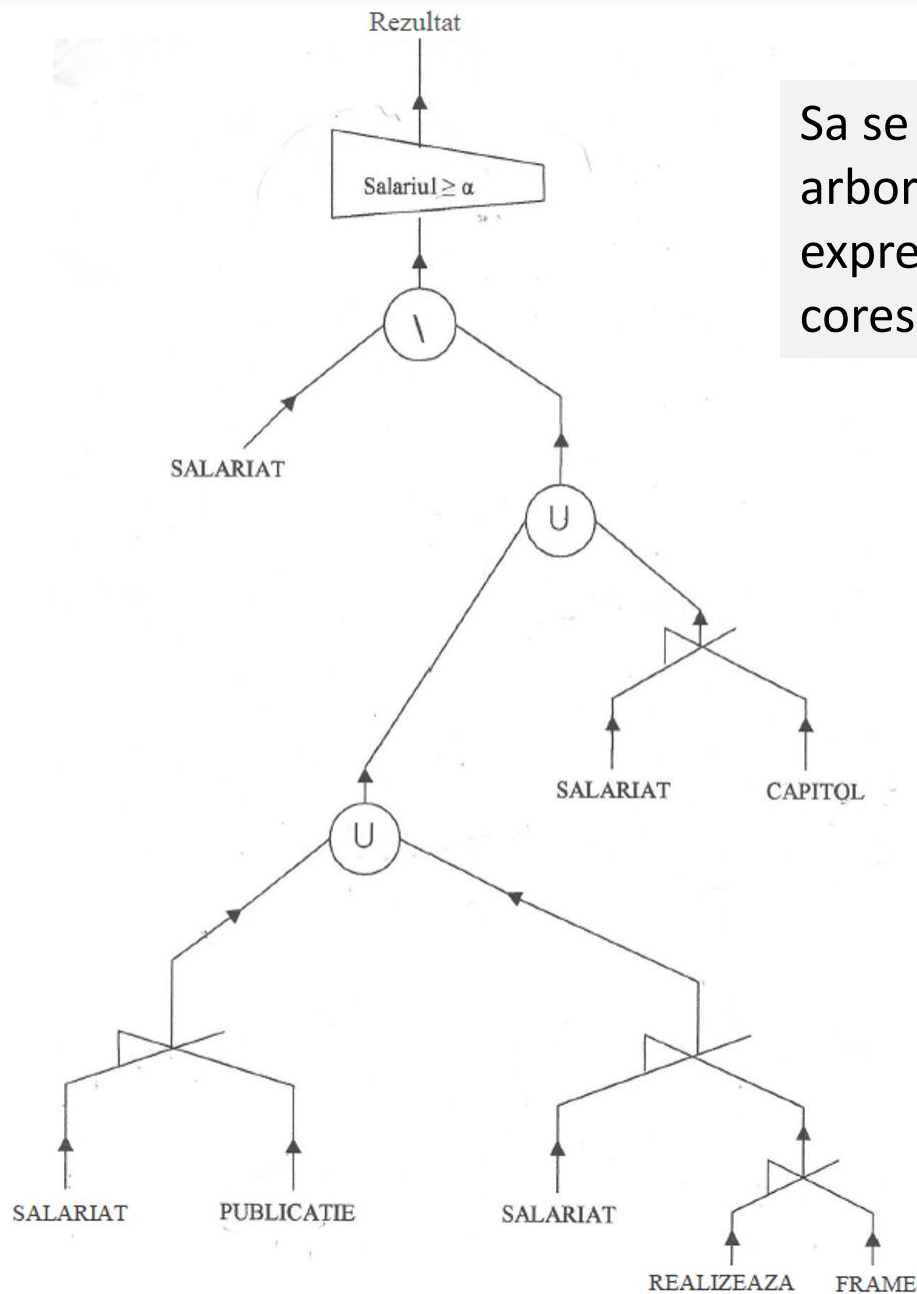
Exercitii Arbori

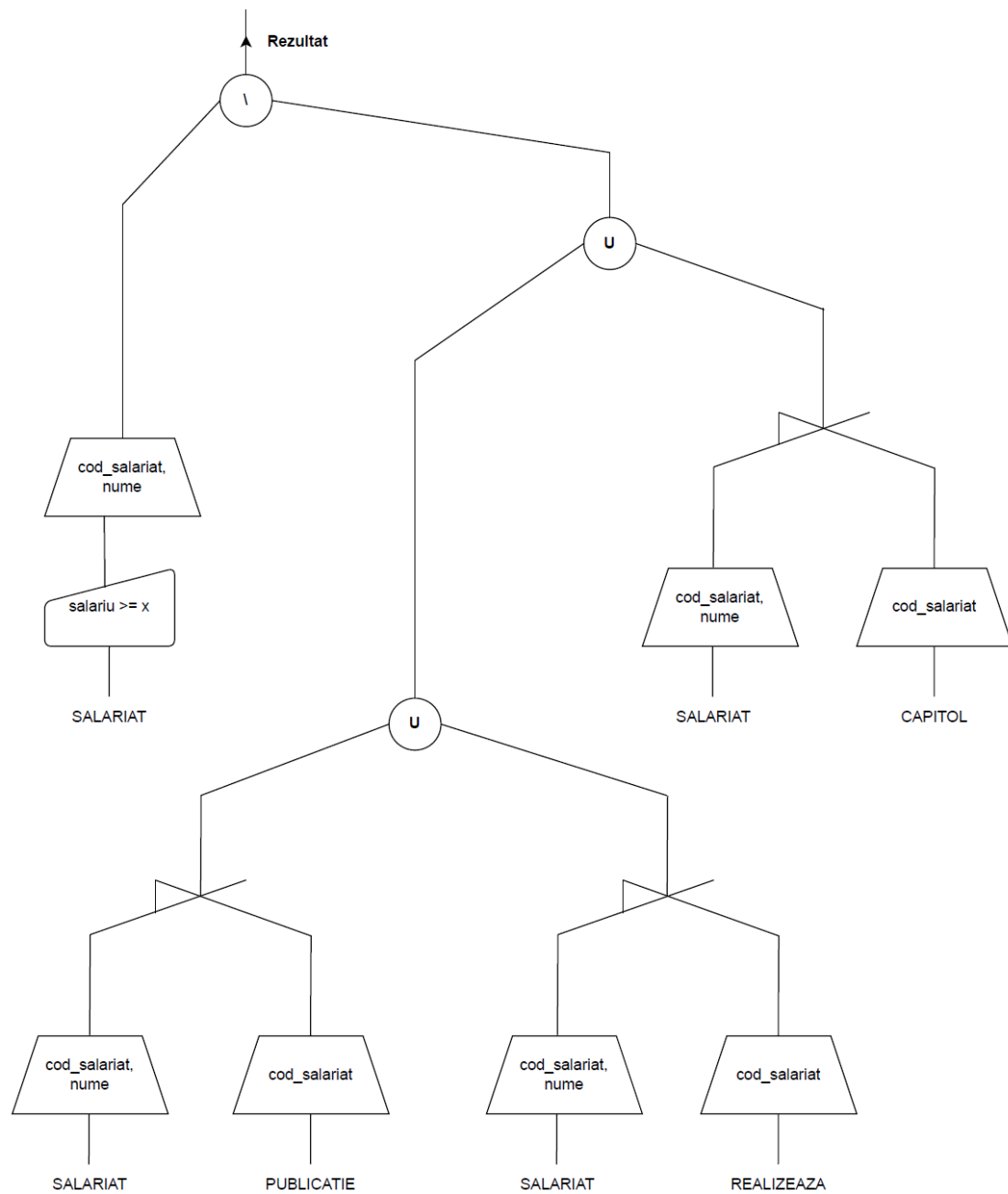
Rezolvarile exercitiilor din cadrul acestui curs se afla si pe site in documentul numit “**Exercitii_Abori**”

Exercitiul 1

Sa se afiseze informatii despre salariatii care nu contribuie la machetarea niciunei publicatii, dar au salariul mai mare decat o valoare data – **cerere in limbaj natural**







Expresia algebrica:

R1 = PROJECT(REALIZEAZA, cod_salariat)

R2 = PROJECT(SALARIAT, cod_salariat, nume)

R3 = SEMIJOIN(R1, R2, cod_salariat)

R4 = PROJECT(PUBLICATIE, cod_salariat)

R5 = SEMIJOIN(R4, R2, cod_salariat)

R6 = UNION(R3, R5)

R7 = PROJECT(CAPITOL, cod_salariat)

R8 = SEMIJOIN(R7, R2, cod_salariat)

R9 = UNION(R8, R6)

R10 = SELECT(SALARIAT, salariu >= x)

R11 = PROJECT(R10, cod_salariat, nume)

R12 = MINUS(R11, R9)

Rezultat = PROJECT(R12, cod_salariat, nume)

Exercitiul 2

Codurile si numerele de telefon ale investitorilor, impreuna cu valoarea investitiei si durata de executie a lucrarilor pentru care valoarea investitiei este intre doua limite **I1** si **I2**, iar tipul contractantului este '**investitie**'.



Scriti **cererea SQL**, urmata de **expresia algebrica** si **arborele algebric**.

Codurile si numerele de telefon ale investitorilor, impreuna cu valoarea investitiei si durata de executie a lucrarilor pentru care valoarea investitiei este intre doua limite **I1** si **I2**, iar tipul contractantului este '**investitie**'.



Cerere SQL?

Codurile si numerele de telefon ale investitorilor, impreuna cu valoarea investitiei si durata de executie a lucrarilor pentru care valoarea investitiei este intre doua limite **I1** si **I2**, iar tipul contractantului este **'investitie'**.



```
SELECT cod_contractant, telefon, val_investitie, durata  
  
FROM contractant co JOIN contract ct ON(co.cod_contractant =  
ct.cod_contractant)  
  
WHERE val_investitie > I1 AND val_investitie < I2 AND  
  
tip_contractant = 'investitie';
```

```
SELECT cod_contractant, telefon, val_investitie, durata  
  
FROM contractant co JOIN contract ct ON(co.cod_contractant =  
ct.cod_contractant)  
  
WHERE val_investitie > l1 AND val_investitie < l2 AND  
  
tip_contractant = 'investitie';
```

Expresia algebrica - neoptima:

R1 = PROJECT(CONTRACT, cod_contractant, val_investitie, durata)

R2 = SELECT(R1, val_investitie > l1)

R3 = SELECT(R2, val_investitie < l2)

R4 = PROJECT(CONTRACTANT, cod_contractant, telefon,
tip_contractant)

R5 = SELECT(R4, tip_contractant = 'investitie')

R6 = PROJECT(R5, cod_contractant, telefon)

Rezultat = JOIN(R6, R3, cod_contractant)

Expresia algebrica:

$R1 = \text{PROJECT}(\text{CONTRACT}, \text{cod_contractant}, \text{val_investitie}, \text{durata})$

$R2 = \text{SELECT}(R1, \text{val_investitie} > 11)$

$R3 = \text{SELECT}(R2, \text{val_investitie} < 12)$

$R4 = \text{PROJECT}(\text{CONTRACTANT}, \text{cod_contractant}, \text{telefon}, \text{tip_contractant})$

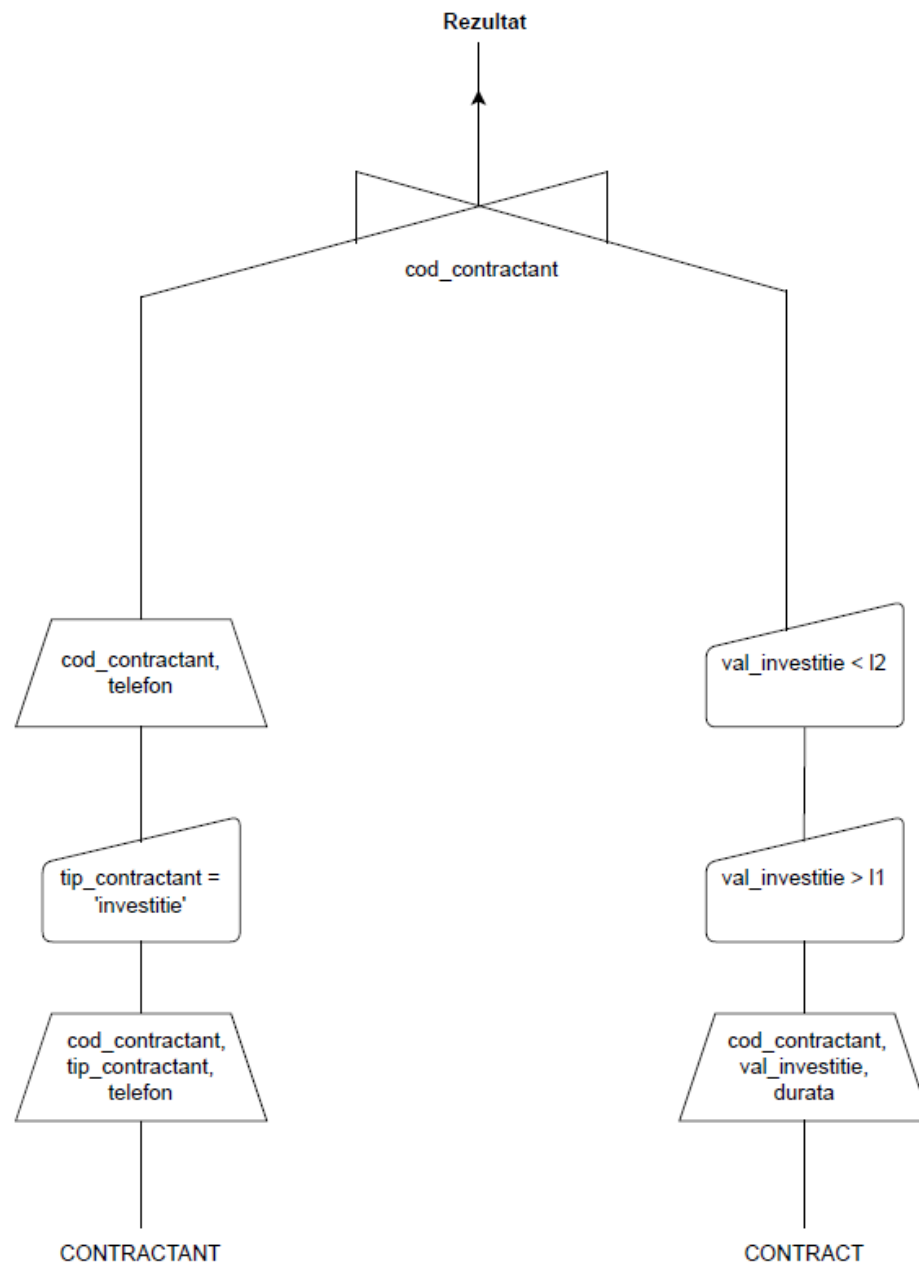
$R5 = \text{SELECT}(R4, \text{tip_contractant} = \text{'investitie'})$

$R6 = \text{PROJECT}(R5, \text{cod_contractant}, \text{telefon})$

Rezultat = $\text{JOIN}(R6, R3, \text{cod_contractant})$

Join-ul din relatia **Rezultat** returneaza toate coloanele pe care le au relatiile primite ca argument.

Arborele algebric - neoptimizat:



Expresia algebrica - neoptimizata:

R1 = PROJECT(CONTRACT, cod_contractant, val_investitie, durata)

R2 = SELECT(R1, val_investitie > 11)

R3 = SELECT(R2, val_investitie < 12)

R4 = PROJECT(CONTRACTANT, cod_contractant, telefon, tip_contractant)

R5 = SELECT(R4, tip_contractant = 'investitie')

R6 = PROJECT(R5, cod_contractant, telefon)

Rezultat = JOIN(R6, R3, cod_contractant)

Expresia algebrica - optimizata:


R1 = SELECT(CONTRACT, val_investitie > 11 AND val_investitie < 12)

R2 = PROJECT(R1, cod_contractant, val_investitie, durata)

R3 = SELECT(CONTRACTANT, tip_contractant = 'investitie')

R4 = PROJECT(R3, cod_contractant, telefon)

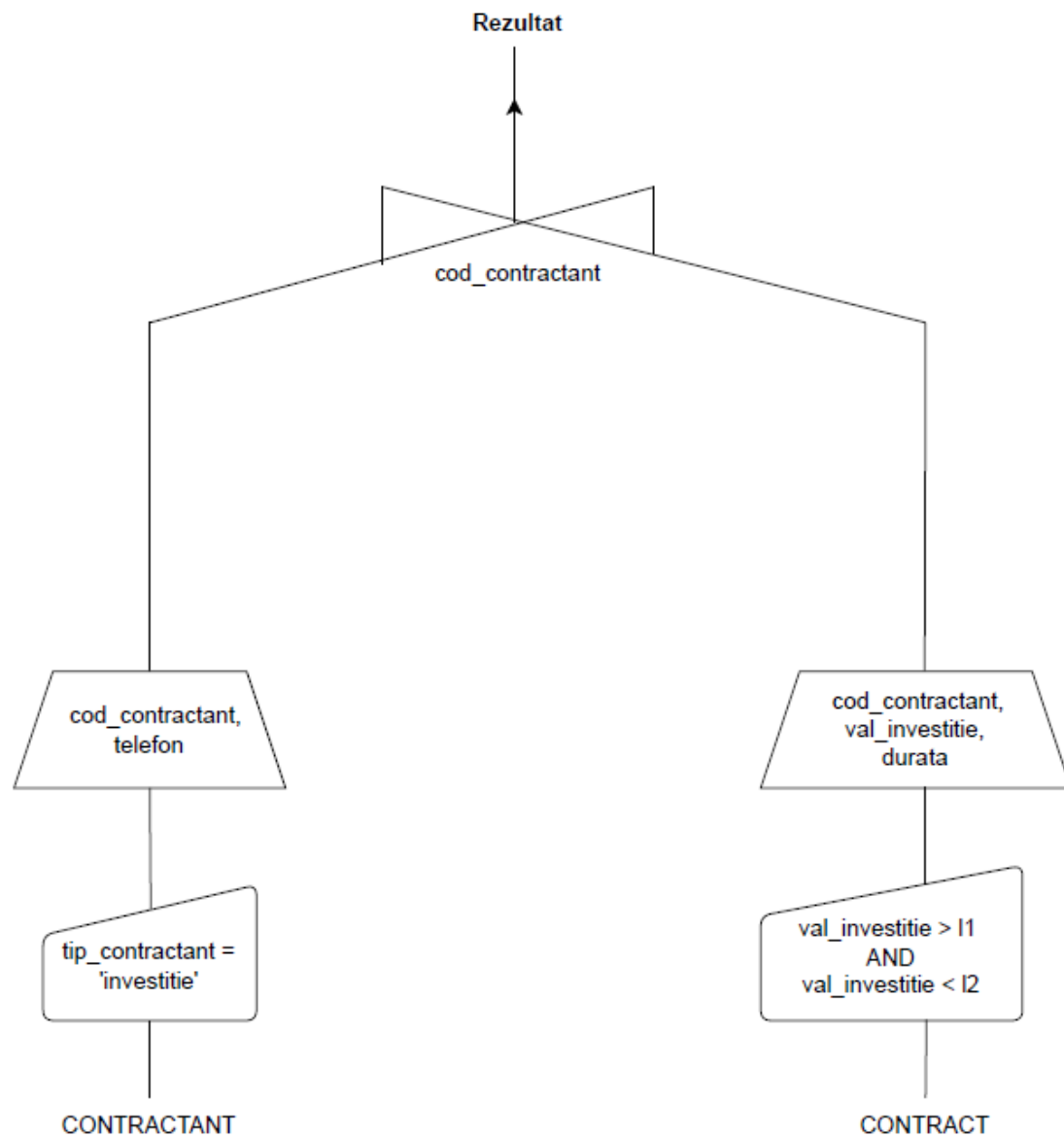
Rezultat = JOIN(R4, R2, cod_contractant) -> rezultat optimizat



Explicatiile detaliate si pasii optimizarii se afla in documentul **Exercitii_Arbori** de pe site.

In acelasi mod se procedeaza si in cadrul proiectului final.

Arborele algebric – optimizat:



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 **12 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 reprezentării informației.** *Într-o bază de date relațională, informația este reprezentată la nivel logic sub forma unor **tabele ce poartă numele de relații**.*
- **Regula 2 – regula accesului garantat la date.** *Fiecare valoare dintr-o bază de date relațională trebuie să poată fi accesată în mod logic printr-o **combinație formată din numele relației, valoarea cheii primare și numele atributului**.*
- **Regula 3 – regula reprezentării informației necunoscute.** *Un sistem relațional trebuie să permită utilizatorului definirea unui tip de date numit „**null**” pentru reprezentarea unei informații necunoscute la momentul respectiv.*

Regulile lui Codd

- **Regula 4 – regula dicționarilor 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 5 – regula limbajului de interogare.** *Trebuie să existe cel puțin un **limbaj pentru prelucrarea bazei de date**.*
- **Regula 6 – 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 7 – regula limbajului de nivel înalt.** *Regulile de prelucrare asupra unei relații luate ca întreg sunt valabile atât pentru operațiile de regăsire a datelor, cât și asupra operațiilor de inserare, actualizare și ștergere a datelor.*

Regulile lui Codd

- **Regula 8 – 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 9 – regula independenței logice a datelor.** *Programele de aplicație trebuie să fie transparente la modificările de orice tip efectuate asupra datelor.*
- **Regula 10 – regula independenței datelor privind integritatea acestora.** *Regulile de integritate trebuie să fie definite într-un sublimbaj relațional, nu în programul de aplicație.*

Regulile lui Codd

- **Regula 11 – 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 12 – 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 relational**:

- 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 satisface în 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**, adică obținerea de relații „moleculare” fără a pierde nimic din informație, pentru a elimina:
- **redundanța;**
 - **anomaliile 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 dependență** nedorită, care este transformată în dependență admisibilă, adică dependență 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 schema;

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 attributele 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 attribute;

NORMALIZAREA RELAȚIILOR

- **Schema sintezei** pleacă de la o mulțime de attribute independente.
 - Utilizând proprietăți de semantică și legături între attribute 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 dintre attribute;

Dependențe funcționale

- O **relație universală** este o relație ce grupează toate atributele care modelează sistemul real cercetat
- Fie E , **mulțimea dependențelor** considerate de proiectantul bazei de date 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 .
- Notăție: $X \rightarrow Y$; X este numit **determinant**, iar Y este numit **determinat** (sau dependent).
 - Dependența funcțională $X \rightarrow Y$ este **trivială** dacă $Y \subseteq X$.

Dependențe funcționale

În momentul proiectării și modelării unei baze de date relaționale pot să apară anomalii. Procesul de optimizare a relațiilor conceptuale poartă numele de **normalizare**. Pentru a depista eventualele redundanțe se utilizează **dependențele funcționale**.

Fie $R(A_1, A_2, \dots, A_n)$ o schema relationala.

De exemplu:

EMPLOYEES(employee_id, first_name, salary, job_id, job_title)

emp_id	first_name	salary	job_id	job_title
100	KING	24000	IT_PROG	PROGRAMATOR
101	JOHN	9500	MGR	MANAGER
102	HUNOLD	17000	IT_PROG	PROGRAMATOR

Dependențe funcționale

emp_id	first_name	salary	job_id	job_title
100	KING	24000	IT_PROG	PROGRAMATOR
101	JOHN	9500	MGR	MANAGER
102	HUNOLD	17000	IT_PROG	PROGRAMATOR

Tipuri de anomalii:

- 1) **Redundanța** – stocarea repetată a unor valori (în exemplul anterior IT_PROG și PROGRAMATOR)
- 2) **Anomalii de actualizare** – modificarea valorii redundante de la 1) poate duce la modificarea valorii doar pentru unele înregistrări și nu pentru toate
- 3) **Anomalii de inserare** – nu se poate insera un job (job_id, job_title) decât dacă se inserează în baza de date și un angajat asociat
- 4) **Anomalii de ștergere** – dacă se șterge un angajat se pierde automat și datele despre job

Dependențe funcționale

emp_id	first_name	salary	job_id	job_title
100	KING	24000	IT_PROG	PROGRAMATOR
101	JOHN	9500	MGR	MANAGER
102	HUNOLD	17000	IT_PROG	PROGRAMATOR

Apar aceste anomalii deoarece în relația **EMPLOYEES** sunt attribute care aparțin unor mulțimi diferite X și Y și deci unor relații diferite.

$X \rightarrow Y$ - **X determină funcțional pe Y sau Y depinde funcțional de X**
dacă pentru orice relație **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 (definiția de pe slide-ul 28)

De exemplu:

100 KING 24000 **IT_PROG PROGRAMATOR**
101 JOHN 9500 **MGR MANAGER**

Dependențe funcționale

emp_id	first_name	salary	job_id	job_title
100	KING	24000	IT_PROG	PROGRAMATOR
101	JOHN	9500	MGR	MANAGER
102	HUNOLD	17000	IT_PROG	PROGRAMATOR

De exemplu:

100 KING 24000 **IT_PROG PROGRAMATOR**

101 JOHN 9500 **MGR MANAGER**

IT_PROG determină funcțional **PROGRAMATOR**

Deci o valoare a lui X determină unic o valoare a lui Y. Astfel, se obțin următoarele submulțimi de attribute ale relației EMPLOYEES:

{job_id} -> {job_title} – job_id determina functional pe job_title

Dependențe funcționale

emp_id	first_name	salary	job_id	job_title
100	KING	24000	IT_PROG	PROGRAMATOR
101	JOHN	9500	MGR	MANAGER
102	HUNOLD	17000	IT_PROG	PROGRAMATOR

Se observa urmatoarele submultimi de attribute:

$X' = \{\text{employee_id}, \text{first_name}, \text{salary}, \text{job_id}\}$

$Y' = \{\text{job_id}, \text{job_title}\}$

Din submulțimile X' și Y' de attribute, se obțin următoarele scheme relaționale:

EMPLOYEES(employee_id, first_name, salary, job_id)

JOBS(job_id, job_title)

Dependențe funcționale

emp_id	first_name	salary	job_id	job_title
100	KING	24000	IT_PROG	PROGRAMATOR
101	JOHN	9500	MGR	MANAGER
102	HUNOLD	17000	IT_PROG	PROGRAMATOR

Se observă că informația din relația **EMPLOYEES** trebuie de fapt stocată în două relații, astfel:

EMPLOYEES

emp_id	first_name	salary	Job_id
100	KING	24000	IT_PROG
101	JOHN	9500	MGR
102	HUNOLD	17000	IT_PROG

JOB

job_id	Job_title
IT_PROG	PROGRAMATOR
MGR	MANAGER

Dependențe funcționale

Acest procedeu se numeste **descompunerea relațiilor**.

Descompunerile se realizează pe baza formelor normale pe care le vom studia în cursul următor, iar definirea formelor normale se bazează pe noțiunea de ***dependență funcțională***.

Dependențe funcționale

EMPLOYEES

emp_id	first_name	salary
100	KING	24000
101	JOHN	9500
102	HUNOLD	17000

JOBS

job_id	Job_title
IT_PROG	PROGRAMATOR
MGR	MANAGER

In exemplul anterior exista urmatoarele dependente functionale:

employee_id -> first_name, salary (se citeste: employee_id **determina functional** first_name si salary sau first_name si salary **depind functional** de employee_id)

job_id -> job_title

Dependențe funcționale

Pornind de la aceste dependente putem afirma ca:

- Dacă doi angajati ar avea acelasi `employee_id` inseamna ca este acelasi angajat
- Dacă doi angajati au acelasi `job_id` atunci rezulta ca au implicit si acelasi `job_title`

Dependențe funcționale

- Comparând toate submulțimile de attribute 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 mulțimi de attribute 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 (augmentare).** Pot fi considerate următoarele formulări echivalente pentru această axiomă.
 - Dacă $X \rightarrow Y$ și $X \subseteq Z$, atunci $Z \rightarrow Y$. ($\subseteq \rightarrow$ fiecare element din X este și element în Z ; X este inclus în Z)
 - 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 $X \cup Z \rightarrow Y$.

În urma reuniunii se obțin dependențe funcționale valide

- **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 .
- **Ullman** a demonstrat că axiomele Ax1 – Ax3, numite **axiomele lui Armstrong**, reprezintă o mulțime închisă și completă de axiome.

Dependențe funcționale

- Nu toate dependențele funcționale sunt folositoare 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 (daca $W = \emptyset$);
 - reflexivitatea nu poate fi utilizată pentru a obține dependențe totale;

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. Multimea tuturor dependențelor funcționale care se pot deduce din F se numeste închiderea mulțimii de dependente F , notata cu F^+
- Două mulțimi de dependențe funcționale totale sunt **echivalente** dacă au **închideri pseudo-tranzitive identice**

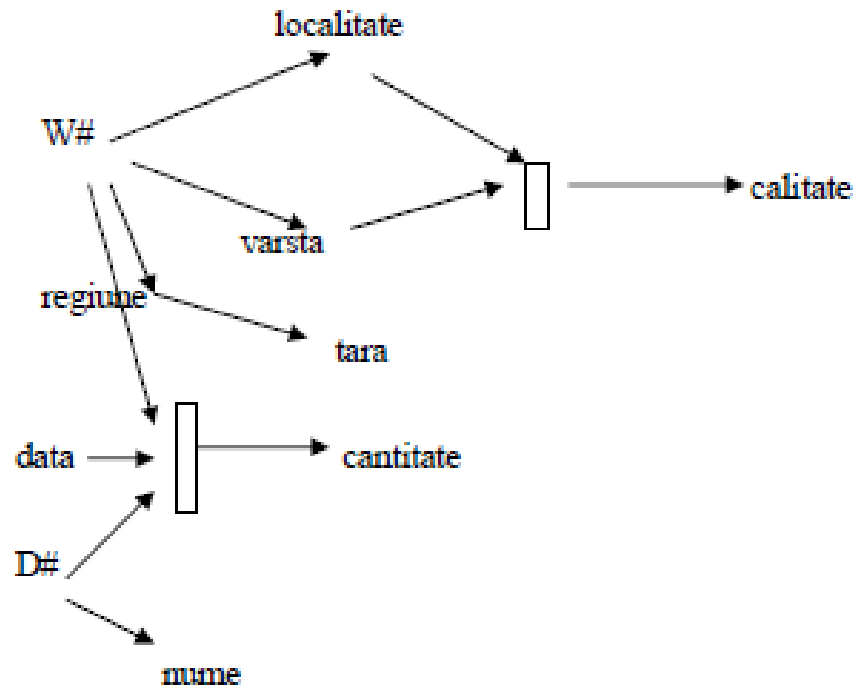
Dependențe funcționale

- Dependențele funcționale între attributele 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).



Dependențe funcționale

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

