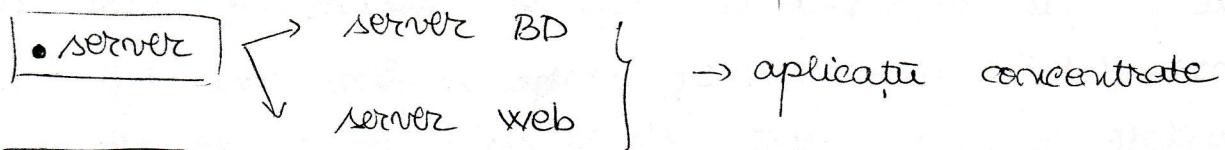


Curs

sapt 13

Aplicații de Bază de Date Web

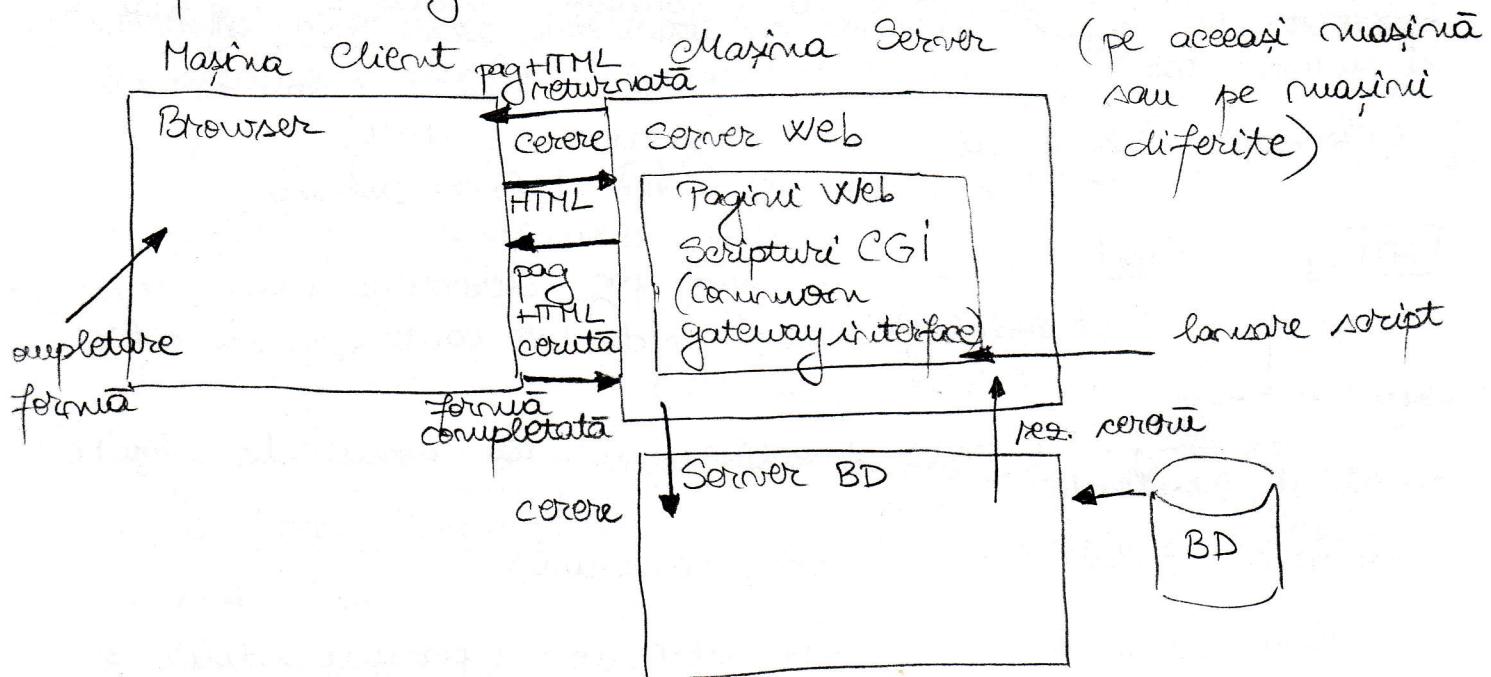
→ sunt arhitecturi client-server particulare



• **client** - aplicație trivială (browser) (nu trebuie să nuște d.p.v. al clientului, totul se desfășoară la nivel de server)

Avantaje: întreținerea este mult mai ușoară decât cele clasice ; partea de client înglobată la nivelul serverului

Web (update numai la nivelul serverului este suficientă, D.p.v. tehnologie).



→ browser interpretează pag HTML de pe serverul web (cerere HTML este rez de server primind întoarcerea pag HTML cerută) (pag HTML - pag statică sau dinamică)

→ după ce se completează datele în formă și se trimite către server va avea în ACTION lansarea unui script ; acesta culege datele din formă și trimite cererile formulate către serverul de BD , rez cerere este transmis serverului web , care trimite apoi o pag HTML către browser.

Componente ce trebuie dezvoltate:

① **pag HTML** - stocate de serverul web

② **scripturi CGI** (scrise în diverse limbaje) = este un program executabil ce este compilat într-un modul de programare (poate fi scris în VB, C, VC etc.); pt a fi script trebuie să respecte o serie de restricții ref. la modul în care scriptul captează datele din formă (culege informații din pag HTML trimisă de către client), trebuie să aibă o acțiune (formulează cererea captată din formă către serverul de BD - precedată de conectarea la server BD, etc. scriptul culege rez transmisie de serverul de BD și incarea rez în pag HTML se sunt transmise browserului și răbășă pe mașina client.)

③ **serverul de BD**

HTML - limbaj de adnotare, folosește directive (`<tag-uri>` ; `<nume-tag>` ; este specializat în afișarea / vizualizarea informație (XML - descrierea structură de date) pt transferul BD între diferite aplicații

HTML → head (titlu pag, nume pag, directive pass, directive index; specif. unde să caute, ce să afiș, inform. generică)

body (toate directivele arată modul de afișare)
end body

→ Zona statică (poze, text, link-uri)

Zona dinamică `<FORM attribute>` (permite utilizarea interacț. direct cu pag; aici se scriu cererile)

`< FORM ACTION = "url" METHOD = "POST | GET" >`
| continut formă

`< IFORM >`

(intotdeauna se găsește în BODY)

"url" - adr. relativă / absolută a unui script CGI

Sensul **ACTION** este de a indica scriptul CGI în modu
lu în care se face submit (lansarea scriptului în execuție)

Adr relativă = dacă scriptul CGI se află în același director

Adr absolută = dacă scriptul CGI se află pe altă mașină

METHOD = indică modul în care scriptul culege informații
din formă ; metoda **DEFAULT** este metoda **GET** (informații din
formă sunt transmise scriptului prin trăsătură de mediu
QUERY STRING sub forma Simbol1 = Valoare1 & Simbol2 =
valoare2 ...

Simbol = numele corespondentă dinamice din pag HTML (specif
în directiva INPUT /SELECT sau TEXTAREA ; numeul obiectului
de acest tip)

Valoare = conținutul introdus de utilizator în cadrul respectiv

Metoda **POST**: este utilizată în general când ur perechilor
(simbol⇒valoare) este mare , iar transmiterea acestora către
script se face prin fizicul standard de intrare (ex: C:\stolin
directive din conținutul FORM:

1. **<INPUT TYPE = "tip" NAME = "nume_simbol" ... >**

optionale depind
de tip

(crează un box pt a introduce informație pe o singură linie
de ex: **type = "text"** => se introduce text pe o
singură linie)

↓
formă liniată

TYPE = "Radio" => butoane de tip radio: (ex: pt.
a genera 3 butoane radio directiva INPUT se utilizează
3 ori partajează același nume , dar val diferență în fct. de
cel care este apăsat)

Ex: Studiu: Mediu Superioare Doctorat

<INPUT TYPE = "RADIO" NAME = "Stud" VALUE = "1"> Mediu

<INPUT TYPE = "RADIO" NAME = "Stud" VALUE = "2"> Superioare

dépende de val atributului WRAP = 'HARD' / 'SOFT' / 'OFF'
 (Soft = informație binarizată, Hard = informație pe linii, Off = informație care nu a fost introdusă)

OBS: În afara directivei <FORM> se pot utiliza aceste <END FORM>

directive deoarece se utilizează de multe FRAME-uri, fiecare FRAME conținând un singur <FORM>.

Base de Date Distribuite

La proiectarea unei BD aveți de ales între ① BD concentrată (stocată pe o singură masină și utilizată un singur server, avantaj = menținerea consistenței BD); ② BD distribuită (împărțire pe mai multe mașini, fiecare cu sătul un server de BD)

Operațiile din BD distribuite se împart în urmă categorii:

1) monogenie = pe mașinile pe care sunt stocate componentele se utiliză același ENGINE de BD (DBMS)

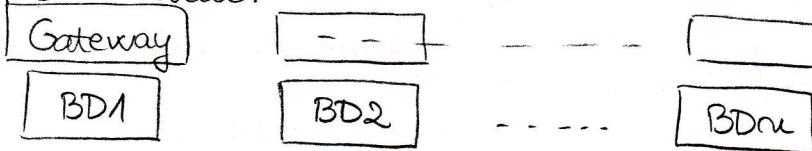
2) eterogenie =

- II -

având ENGINE (mediu de BD) diferenți.

comunicare

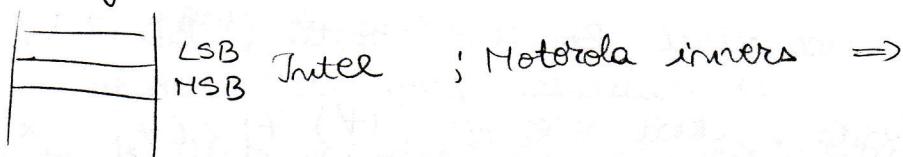
Structural:



asigură legătura între ENGINE de BD și infrastructura de comunicare
 (cpl. soft)

OBS: să adapteze formatul datelor corespunzătorului se rulează pe mașina resp. cu formatul datelor aferent mediului de la care s-a făcut interog sau spec. care se transmite BD (gateway; pt BD monogenie/ eterogenie)

OBS:



BD proiectată trebuie distribuită pe mai multe mașini.
Distribuția se face după o anumită logică a.i. să se evite transferul masiv de date

$$BD \text{ concentrată} = (\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_k)$$

Fragmente ale BD; poate fi fragmentată

prin a) fragmentare orizontală (o relație este descompusă în 2 fragmente având aceeași componență) ex: Angajat Salariu

Fragment pe orizontală: Ang1 (.... Salariu....)

<500

Ang2 (.... Salariu....)

>500

(dintr-o tabelă înreg care îndeplinește anumită condiție sunt fixate pe un site, restul pe alte site-uri)

b) fragmentare verticală (fragmente contin componențe distincte, eventual cu componență cheie comună)

ex: Ang1 (cnp, nume, prenume, salar)

Ang2 (cnp, adr, funcția, ...)

a) refacerea tabelei inițiale prin reunire

b) -II- prin JOIN (cu condiție de JOIN

componență cheie identică pt. tabelele impărțite)

Problema distribuției:

$$BD = (\mathcal{F}_1, \dots, \mathcal{F}_n)$$

$$S = (S_1, S_2, \dots, S_k)$$
 site-uri

Alegerea site-urilor pe care vor fi distribuite fragmentele
Distribuția trebuie să satisfacă unele cerințe:

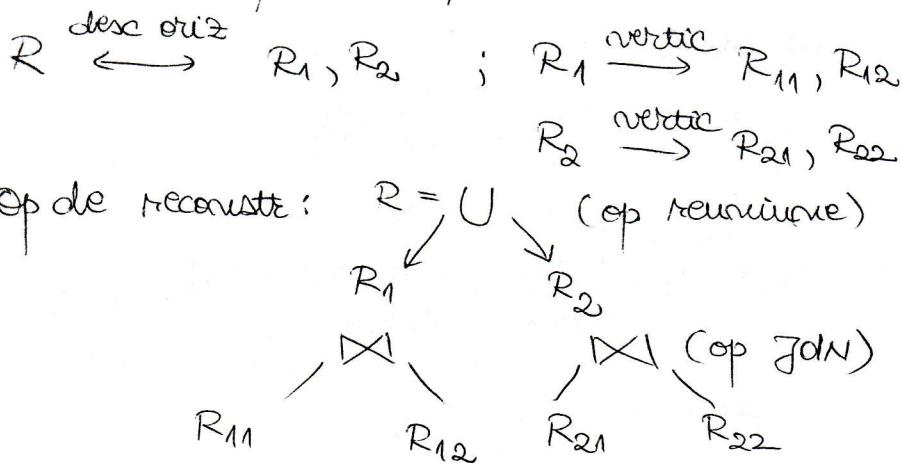
1) Consistență: $\forall x \in BD \quad (\exists i \text{ a.i. } x \in \mathcal{F}_i)$

2) Reconstrucția: există un operator aplicat fragmentelor prin care putem reconstrui BD concentrată; $BD = \nabla(\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_n)$

3) Redundanță: dacă $x \in \mathcal{F}_i \rightarrow (\nexists j, i \neq j \text{ s.t. } x \notin \mathcal{F}_j)$

Cerința garantă într-un fragment nu se regăsește în alte fragmente

Se aplică la fragm. oriz; la fragm. vertic. (J) cel puțin 7.
redundanță pt. compusul dreic.



(ordinea operat. depinde de tipul de fragmentare)

ex: Fragm pe orizontală:

$F_1 = \text{Ang}_1(\text{ang}, \dots) \text{ Salariu} < 500$

$F_2 = \text{Ang}_2(\dots) \text{ Salariu} \geq 500$

Presup că: 1) să se găsească angajii cu $\text{Sal} \in [300; 700]$

Angajat : $S_{<\text{Salariu} > 300 \text{ and } \text{Salariu} < 700 >}^{(\text{Angajat})}$

Fragm : $R_1 = S_{<\text{Salariu} > 300 >}^{(\text{Ang}_1)}$

$R_2 = S_{<\text{Salariu} < 700 >}^{(\text{Ang}_2)}$

$$R_{\text{rez}} = R_1 \cup R_2$$

Cererea a fost descompusă în 2 cereri care au fost ruleate pe masini separate și s-a facut reunirea celor 2 rez.

2) $\text{Salariu} > 600$

$S_{<\text{Salariu} > 600 >}^{(\text{Angajat})}$

$R_{\text{rez}} = S_{<\text{Salariu} > 600 >}^{(\text{Ang}_2)}$ (cererea este executată pe un singur site)

Catalogul distribuit al BD CD = sita localizarea și modul de distribuție al BD.

SELECT * FROM Angajat

WHERE Salariu > 300 and Salariu < 700

Optimizatorul SQL constă în formulează una sau mai multe cereri:

(SELECT *
FROM Angajat → site S1
WHERE Salariu > 300) UNION

(SELECT *
 FROM Angajat
 WHERE Salariu < 700) → site S₂ > agregare U cu UNION

3) BD → Angajat fragă ca mai sus;

SELECT AVG(Salariu)
 FROM Angajat

! SELECT AVG(Salariu)
 FROM Angajat

NU obținem rez correct
 SELECT AVG(Salariu)
 FROM Angajat

→ S₁

→ S₂

SELECT SUM(Salariu), COUNT(*)
 FROM Angajat

→ S₁

SELECT SUM(Salariu), COUNT(*) → S₂
 FROM Angajat

SUM₁ + SUM₂

COUNT₁ + COUNT₂

Cererea este preluată de ENGINE-ul de distribuție ce cunoaște CD; descompunere cerere și refacere rez. nu prezintă utilizatorul, este scopul ENGINE-ului de distribuție.

↓
 scrie cererea SQL ca și cum aru avea BD concentrată,

-ex:

F₁ Jasi

F₂ București

Presup că Jasi: tabela Angajat

Buc: tabela Departament

nu (Angajat) nu (Departament) nu > me

Să se afișeze numele și prenumele managerilor.

SELECT nume, prenume

FROM Angajat, Departament

WHERE crnp = Manager

unit de măs. ident

Notăm că D = tipul de citire/scriere pt o pagină

(volum de informații ex: 4KB pt că record Ang ≠ record Depart)

s = tip transport pt pagină

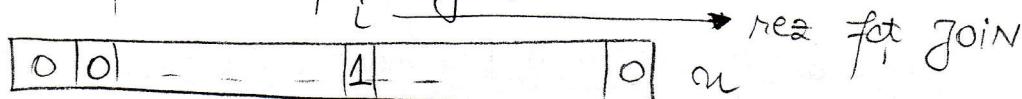
Angajat necesită 10^3 pagini

Departament → 50 pagini

Execluție la București: $10^3(D+S) + 50 \cdot D = 1050D + 1000S$

Taxi: $50(D+S) + 10^3D = 1050D + 50S$

Se rez că **BLUM join** se referă la construirea unui vector cu val booleenii constr. pe baza funcției HASH; vectorul T_b să aibă atâtăa componente cât val posibile întoarse fct. HASH. Acolo unde la sănăpul din tabelă la care se aplică HASH se obține o val inv într vectorului pe cele n poz. Se setează bitul coresp poz. val întoars de fct HASH, obținând în cazul de față cu multe valori nule; la site-ul coresp se transmite acest vector se calc aceeași fct HASH pe sănăpul după care se face JOIN; dacă pt o invreg se a întors o anumită val bitul în vector este setat \Rightarrow invreg îndepl cond de join; dacă pt poz vect întoars de rez fct. HASH se găsește $\emptyset \Rightarrow$ invreg nu îndepl cond de join; în final se poartăaza din tabelă coresp doar acel invreg de au îndepl JOIN pt fct hash, urmând ca acele invreg să fie transmisă pt a se face JOIN.



Transmit vect la site-ul coresp; se aplică aceeași fct HASH; dacă e 1 \Rightarrow îndepl cond JOIN

\rightarrow fct HASH rezist. la coliziuni - problema cea mai importantă este găsirea acestei fct HASH

ex: aplicată în rezervare (mai ales a biletelor de avion)