

# Baze de Date

## Ingineria sistemelor an III

### Prof. dr. ing. Dorin Carstoiu

## Biliografie

1. A. Ullman, DataBase Design, 1972
2. D. Carstoiu, Tehnologia bazelor de date, Lit. UPB, 1992
3. D. Carstoiu, Baze de date relationale, Ed Printech, 1999
4. F. Radulescu, Baze de date Internet, Ed. Printech, 2000
5. F. Radulescu, PL1, Ed. Printech, 2005
6. D. Carstoiu si altii, DB2, Ed. Agir 2007
7. D. Carstoiu, Baze de date, Ed Matrix ROM, 2009
8. D. Carstoiu, A. Olteanu, A. Cernian Baze de date – Indrumar de laborator, Ed Matrixrom, 2009.
9. \*\*\* BigTable, Hbase, MongoDb, Cassandra
10. \*\*\* Oracle PL1
11. \*\*\* Microsoft SQL
12. \*\*\* MySql, PHP.

# Definitii

- *O baza de date* este un ansamblu de date structurate stocate astfel incat sa faciliteze accesul unui calculator, ce asigura accesul simultan a mai multor utilizatori, capabila sa raspunda la cererile acestora in timp util (**se comenteaza semnificatia fiecarei asertiiuni**).
- Un *Sistem de Gestiune al Bazei de Date (SGBD) (engleza: DataBase Management System - DBMS)* este format din totalitatea programelor ce permit definirea, manipularea, vizualizarea, stocarea si gestionarea interogarilor pentru accesul la date.

## Componentele unei aplicatii baze de date

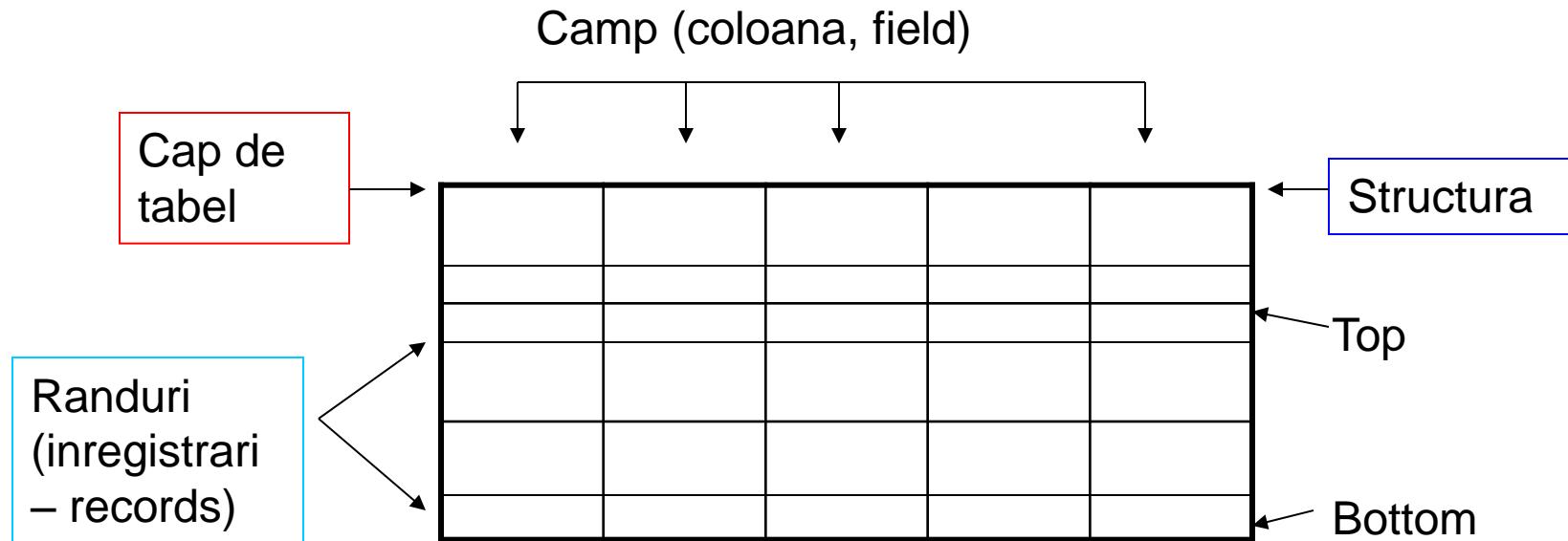
- Baza de date propriuza
- Programe de aplicatie

Proiectarea aplicatiilor necesita respectarea seveniei: proiectarea bazei de date si apoi a programelor de aplicatii ce asigura viziunea asupra bazei de date din diverse perspective.

- Oricare ar fi definitia o baza de date are cateva caracteristici implice:
- O baza de date formeaza o *colectie logica coerenta* de date ce are cel putin un intelese.
- Baza de date este destinata, construita, si populata cu date dintr-un domeniu bine precizat. Ea se adreseaza inerent unui anumit grup de utilizatori ce folosesc un grup specific de aplicatii.
- O baza de date modeleaza cteva aspecte ale lumii reale creand ceea ce se numeste *orizontul propriu*, orice modificare a orizontului fiind reflectata in baza de date.
- O baza de date poate fi de diverse marimi si complexitati. De exemplu, lista numelor, adreselor si numerelor de telefon pastrate de o persoana particulara poate avea sute de inregistrari fiecare cu o structura simpla. Pe de alta parte catalogul cartilor intr-o mare biblioteca poate contine milioane de inregistrari ce pot fi vazute in diverse organizari din punctul de vedere al ordonarii, dupa numele primului autor, dupa subiect sau dupa titlul cartii.
- O baza de date poate fi generata si intretinuta manual sau cu ajutorul unor programe utilitare specializate.

# Tabela, obiect al bazei de date

O baza de date este organizata ca o colectie de tabele, similar cu tabelele pastrate in format tiparit, dar si alte obiecte.



Orice tabela are asociat un nume si nu este intotdeauna vazuta ca un fisier secvential.

# Structura unei tabele

Prin definirea structurii unei tabele se intlege stabilirea numelui campurilor, a tipurilor de date asociate si a formatelor asociate.

**Obs. Foarte importante sunt constrangerile asociate tabelelor.**

- **Nume camp** - sir de caractere, similar cu numele variabilelor in limbajele de programare, limitat ca lungime si caractere permise functie de mediul de baze de date
  - **Tip data, format** – tip generic functie de numarul de octeti necesari pentru stocare (defineste mai mult formatul de afisare nu cel de stocare)

**OBS.** Tipurile definite sunt tipuri generice, ele difera de la un mediu la altul (char(n), varchar(n), string, integer(n), number (m.n), float, real, date etc)

## Strucura tabela:

Nume\_tabela (nume\_camp1 tip\_data [format],  
nume\_camp2 tip\_data [format],  
.....)

## OBS:

- Campurile pot avea o serie de atribute suplimentare privind restrictiile de integritate (not null, primary key, foreign key etc).
- Campurile au asociata ordine fizica identica cu ordinea de definire.
- Fiecare camp este vazut ca o variabila careia ii este atribuita valoarea corespunzatoare din inregistrarea curenta.
- Fiecare tabela are asociat un **indicator de inregistrare**, care indica inregistrarea curenta.
- Navigarea intr-o tabela de la o inregistrare la alta are ca efect actualizarea valorii in “variabila” nume\_camp
- O baza de date este formata dintr-o colectie de tabele intre care exista o serie de asocieri sau relatii:

Ex: Mybase(nume\_tabela1, nume\_tabela2, ...., nume\_tabelan)

# Clasificare SGBD – criterii nestandard

## 1. Dupa modelul datelor

- Model relational (peste 90%) -bazat pe modelul entitate asociere
- Model ierarhizat (arbore)
- Model retea (graf orientat)

## 2. Dupa localizarea datelor

- Baze de date centralizate
- Baze de date distribuite (model SQL sau non SQL)

## 3. Dupa complexitatea bazei de date manipulate

- SGBD comerciale (pentru baze de date de volum rezonabil, fara instrumente de administrare puternice) Fox, Access, MySQL ...
- SGBD profesionale (capabile sa manipuleze baze de date de mari dimensiuni, securitate ridicata) Oracle, Progress, DB2...

## 4. Dupa arhitectura aplicatiei suportate nativ:

- SGBD monouser (nu permite acces partajat la aceeasi resursa sau este permisa prin alte mecanisme)
- Medii multiuser

5. Dupa limbajul de dezvoltare a aplicatiilor:

- Limbaj nativ (propriu SGBD)
- Limbaj gazda (doar engine SGBD, aplicatii dezvoltate in alte limbaje suportate)

6. Dupa limbajul de implementare cereri:

- Limbaje SQL (Structured Query Language) – raspandire mare;
- Limbaje QBE (Query By Exemple) – ex: Paradox;
- Limbaje 4GL – Progress

7. Dupa interfata utilizator:

- Interfata alfanumerica (lucru in consola);
- Interfata grafica (vizuala)

8. Dupa sistemul de operare al calculatoarului pe care se instaleaza:

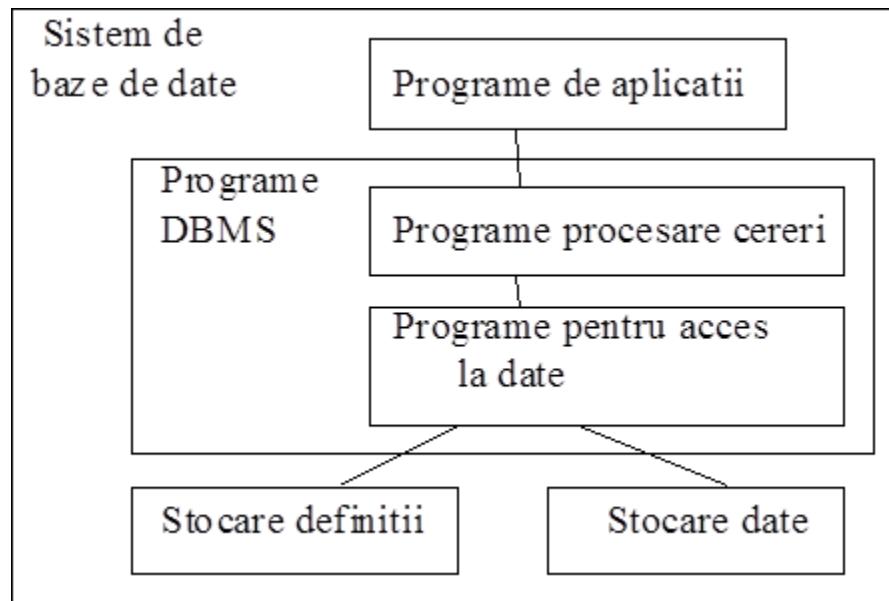
- Distributie pe Windows
- Distributie pe Linux
- Distributie pe Unix
- Distributie pe Mac

**Obs:**

- O parte a SGBD-urilor au distributie pe intreaga gama a sistemelor de operare (ex Oracle), altele numai pe un singur sistem de operare (ex: Microsoft SQL).
- Pot fi gasite si alte criterii de clasificare, cele prezentate urmaresc sa evidenteze o serie de caracteristici ale SGBD.

- *Definirea* bazei de date presupune specificarea structurii si a tipurilor de date ce vor fi stocate in baza de date, precum si descrierea detaliata a fiecarui tip de data.
- *Constructia* bazei de date este procesul popularii si stocarii datelor insasi pe mediul controlat prin DBMS.
- Prin *manipulare* se inteleag o serie de functii ce faciliteaza implementarea cererilor pentru gasirea datelor specificate, adaugarea de noi date ce reflecta modificarea contextului, generarea de rapoarte pe baza continutului bazei de date.
- **Obs.** Se remarcă faptul că nu este imperios necesar un sistem de programe general cum este DBMS sau SGBD, el putind fi înlocuit cu alte programe specifice, caz în care se poate afirma că a fost creat un DBMS specific cu funcții restrânse față de cel universal.
- Reuniunea dintre componentele software ce asigura manipularea datelor, împreună cu datele insasi (continutul bazei de date) formează ceea ce se numește **sistemul de baze de date ( DataBase System )**,

# Sistem de baze de date



- Definitiile sunt stocate in *catalogul sistemului* (*system catalog*) si contine informatii despre structura fiecarei tabele, tipul si formatul de stocare a fiecarei coloane, o serie de restrictii referitoare la valorile posibile ale datelor. Informatiile stocate in catalog mai poarta denumirea de *meta-data*.
- Catalogul este utilizat intern de DBMS si numai in mod ocazional de utilizator atunci cand solicita informatii referitoare la structura bazei de date.
- Ca urmare DBMS nu este destinat unui anumit program de aplicatie specific, el este universal, putand fi utilizat de o multime de programe de aplicatie.
- Catalogul este utilizat pentru a cunoaste structura informatiilor din baza de date specificata, precum si formatul datelor in tabelele componente.
- Acelasi DBMS trebuie sa lucreze la fel de bine in orice categorii de aplicatii: baze de date pentru universitati, banchi, societati comerciale, etc.

# Avantaje baze de date

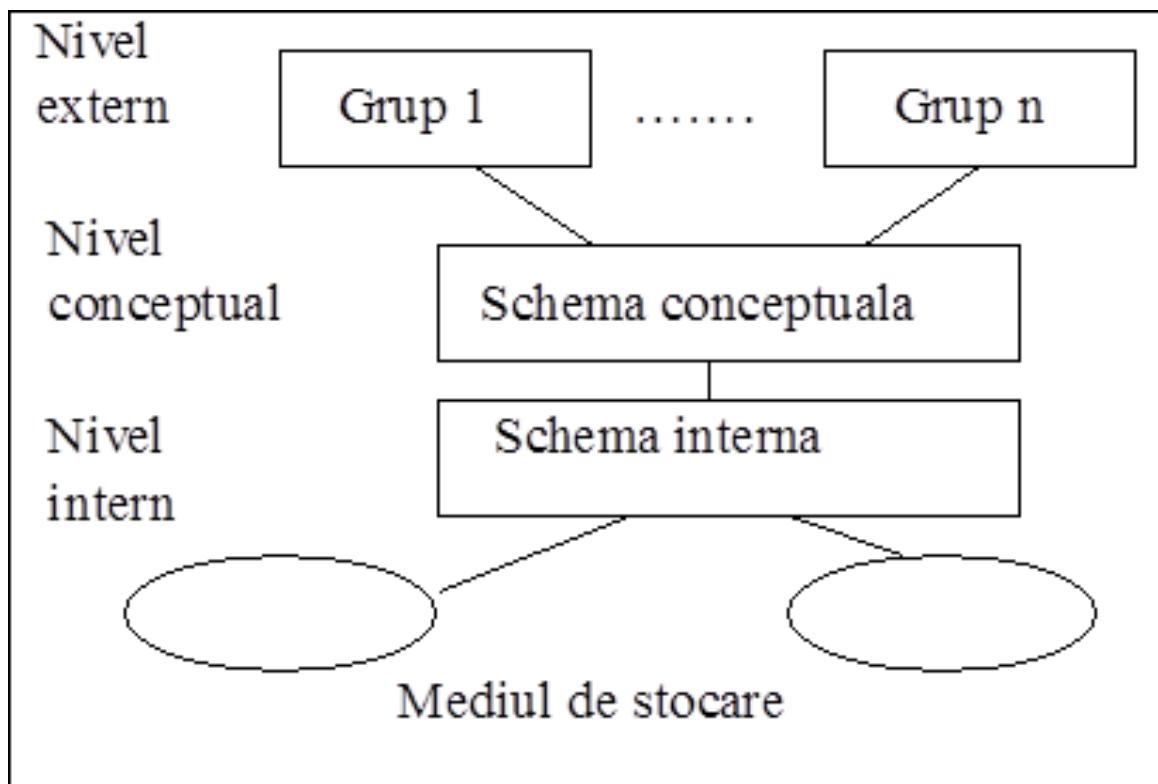
- *facilitarea si grabirea standardizarii* datorita cooperarii largi intre diverse departamente, colective de proiectare, utilizatori. Standardizare se refera la numele si formatul datelor, formatul de afisare, terminologia utilizata, etc.
- *flexibilitatea* consta in posibilitatea modificarii structurii bazei de date fara a fi necesara modificarea programelor de aplicatie.
- *reducerea timpului de dezvoltare a aplicatiilor* analiza, proiectarea si implementarea unei noi aplicatii de baze de date cere mai putin timp decit cel aferent dezvoltarii unor programe particulare capabile sa manipuleze tabele, acesta fiind estimat ca 1/4 pina la 1/6, in principal datorita facilitatilor oferite de DBMS si a interfetei cu utilizatorul foarte prietenoase.
- *facilitatea de reactualizare* prin adaugarea sau stergerea de inregistrari, cat si prin modificarea structurii datelor existente. Daca exista o compatibilitate in ceea ce priveste conversia unui tip de data in alt tip, la modificarea structurii aceasta este facuta automat.
- *economisirea spatiului de stocare* este o consecinta a structurarii datelor prin eliminarea datelor redondante.

# Scheme de baze de date

## *Arhitectura pe niveluri*

- *Nivelul intern* constituit din *schema interna* ce descrie structura de stocare fizica a datelor in baza de date, utilizind un model al datelor fizice. La acest nivel se descriu detaliile complete ale stocarii, precum si modul de acces la date.
- *Nivelul conceptual* sau *schema conceptuala* descrie structura intregii baze de date pentru o comunitate de utilizatori.
- *Nivelul extern* sau nivelul vizual (utilizator), include o colectie de scheme externe ce descriu baza de date prin prisma diferitilor utilizatori. Fiecare grup de utilizatori descrie baza de date prin prisma propriilor interese. Exista tendinta la acest nivel ca grupuri de utilizatori sa ascunda detalii de care nu sunt interesate.

# Modele de date



# Independenta datelor

Organizare pe trei niveluri este foarte importanta pentru ca explica conceptul de independenta a datelor, prin posibilitatea de modificare a sistemului bazei de date la orice nivel fara a avea influenta la nivelurile superioare.

Independenta datelor se defineste de regula in doua moduri, corespunzator nivelelor conceptual si intern.

- Prin *independenta logica* se intlege capacitatea schimbarii schemei conceptuale fara a atrage dupa sine schimbari in schema externa sau in programele de aplicatii.
- *Independenta fizica* este reprezentata prin capacitatea de schimbare a schemei interne fara schimbarea schemei conceptuale sau externe. Schimbarea schemei conceptuale poate surveni ca urmare a reorganizarii fizice a unor fisiere, prin crearea de noi structuri de acces menite sa asigure accesul eficient la date.

# Strucura DataBase Management System (DBMS)

Nucleul central al oricarui mediu de baze de date este DataBase Management System (DBMS) sau SGBD.

DBMS consta dintr-o colectie de programe ce asigura definirea structurilor de date, stocarea datelor, manipularea acestora, vizualizarea, administrarea accesului.

Componente:

1. Data Definition Language (DDL)
2. Storage Definition Language (SDL)
3. View Definition Language (VDL)
4. Data Manipulation Language (DML)
5. DataBase Administrator – Data Control Language (DBA-DCL)

# 1. Data Definition Language (DDL)

DDL este constituit din programe specializate ce permit crearea structurii bazei de date, a structurii tabelelor si a relatiilor dintre acestea.

Caracteristici:

- Prin DDL se genereaza entitatea baza de date si se permite modificarea structurii acestoia, specificarea constraintelor.
- Tipurile de date permise depind de mediul de implementare.
- La operatiile de import/export datele sunt convertite in formatul propriu pentru destinatie.
- De cele mai multe ori nu se garanteaza integritatea datelor la migrarea de la un mediu la altul si chiar migrarea intre versiuni ale aceluiasi SGBD.
- Restrictiile impuse campurilor sunt cele care produc anomalii la migrare.
- Comenzile DDL nu pot fi revocate (se spune ca sunt cu AUTOCOMMIT)

## 2. Storage Definition Language (SDL)

Caracteristici:

- Componenta de regula invazibila utilizatorului
- Este inclusa in nucleul SGBD si asigura modalitatea de stocare a bazei pe memoria externa (exploateaza organizarea pe blocuri).
- Structura de stocare nu asociaza pentru fiecare tabela un fisier.
- Modalitatea de stocare este dependenta de organizarea memoriei externe si de facilitatile de acces oferite.
- Lucreaza in conjunctie cu DBA pentru a restrictiona accesul la date.
- Utilizatorii pot utiliza propriile mecanisme de stocare fara a garanta ca sunt superioare celor folosite de SGBD.
- La bazele de date comerciale componenta lipseste de cele mai multe ori si este suplinita de resurse ale sistemului de operare aferente stocarii de fisiere.

### 3. View Definition Language (VDL)

Caracteristici:

- Destinata manipularii vederilor aferente bazei de date.
- Permite crearea de vederi (views) ca instrumente de facilitare a procesarilor ulterioare.
- Vederile sunt pastrate ca definitii (nu sunt tabele permanente) dar pot fi utilizate similar cu acestea.
- La apelarea unei vederi aceasta este actualizata (calculata)
- Pot fi formulate cereri in care sunt invocate obiecte de tip vederi similar cu invocarea obiectelor de tip tabela.
- Nu toate mediile de baze de date permit definirea de vederi (componenta nu exista).

## 4. Data Manipulation Language (DML)

Caracteristici:

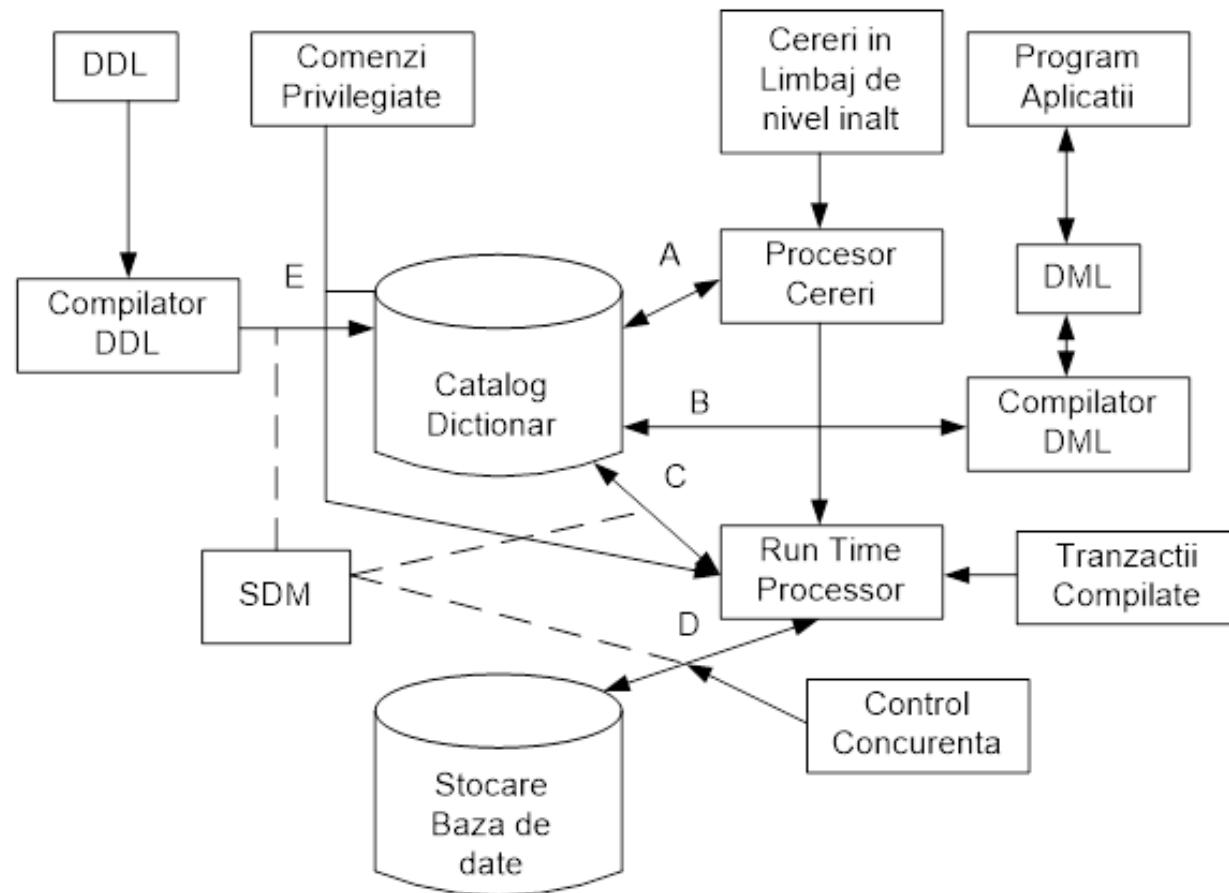
- Componenta esentiala in stabilirea performantelor unui mediu de baze de date
- DML asigura implementarea limbajului de cereri (interrogarea bazei de date).
- Initial, singura componenta standardizata (modele SQL, QBE, 4GL), apoi standardul s-a extins si la alte componente. Standardul poarte denumirea de standard SQL.
- Performantele depind de dimensiunea bazei de date
- Include algoritmi de optimizare a executiei cererilor.
- Are facilitati suplimentare pentru baze de date distribuite (gestiunea tranzactiilor distribuite).
- In baze de date centralizate tranzactiile sunt serializate

## 5. DataBase Administrator (DBA) sau Data Control Language (DCL)

Caracteristici:

- Nu se regaseste in toate SGBD. Cele fara DCL utilizeaza facilitatile sistemului de operare pe care este instalat pentru restrictionarea accesului.
- Asigura stabilirea rolurilor si drepturilor de acces ale utilizatorilor la baza de date
- Implementeaza concepte de securitatea datelor
- Opereaza impreuna cu alte componente SGBD.
- Permite implementarea unei securizari intrinsece suprapusa peste securizarea bazata pe sistemul de operare.

# Schema DBMS



# Caracteristici sisteme de baze de date distribuite

Pentru a implementa un sistem de baze de date distribuite sunt necesare o serie de componente software, dintre care amintim:

- componentele de comunicatie
- DBMS-uri locale
- DBMS distribuit.

DBMS-urile locale sunt cele care descriu si manipuleaza datele din bazele de date stocate pe statiile locale sau altfel numite si noduri de date. Acestea pot fi de acelasi tip sau tipuri diferite, formand ceea ce se numesc sisteme de baze de date omogene sau eterogene.

Prin SGBD distribuit intelegem un sistem software complex care asigura gestionarea bazei de date distribuita pe mai multe noduri ale retelei de calculatoare. Un astfel de SGBD are ca sarcina accesarea datelor in urma cererilor formulate de diferitii utilizatori. El va crea si va pastra la zi dictionarul de date global, dictionar care contine informatii despre baza de date distribuita.

Un SGBD distribuit urmareste atingerea unor obiective specifice gestionarii distribuite a datelor, obiective ce sunt in mare parte comune oricarui SGBD dar cu aspecte specifice datorate caracterului distribuit al datelor.

# Caracteristici sisteme de baze de date distribuite

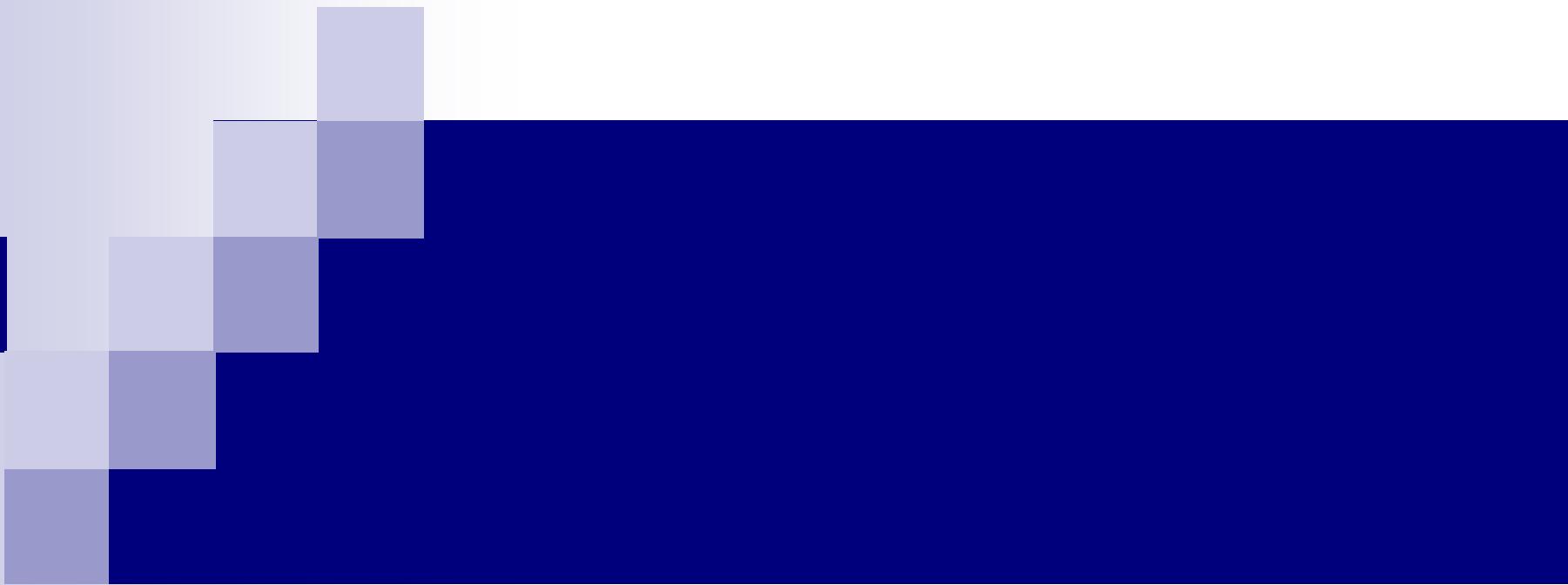
Principalele obiective urmarite:

- *Redundanta minima si controlata* a datelor. Distribuirea datelor in retea poate genera o crestere a redundantei. Exista pericolul de pierdere a coerentei datelor si de crestere exploziva a volumului datelor.
- *Facilitati de utilizare adecvate*. Un SGBD distribuit trebuie sa asigure transparenta localizarii informatiilor pe calculatoarele din retea.
- *Securitatea datelor*. In sistemele ce comunica printr-o infrastructura problema securitatii devine mult mai importanta, atat datorita distribuirii datelor, cat si a existentei unui numar mare de utilizatori care pot accesa datele. Vorbim de securitate locala si securitate globala la nivelul bazei de date distribuite.
- *Coerenta si integritatea datelor*. Sistemele de gestiune a bazei de date locale asigura coerenta si integritatea datelor la nivelul fiecarui nod. La nivelul intregii baze de date distribuite sistemul de gestiune distribuit trebuie sa asigure integritatea datelor in special pentru operatiile de actualizare.
- *Partajarea datelor*. Cum unul dintre scopurile bazelor de date distribuite este cel de acces simultan a mai multor utilizatori la date este foarte importanta localizarea datelor. Sa presupunem scenariul in care un utilizator lanseaza o cerere de regasire de la o statie locala. Cererea este preluata de SGBD distribuit care cu ajutorul dictionarului global de date, localizeaza nodul unde sunt stocate datele necesare.

# Caracteristici sisteme de baze de date distribuite

Situatii posibile functie de partajarea datelor:

- cererea poate fi satisfacuta in intregime pe statia locala de unde ea a fost lansata (cerere locala);
- cererea poate fi satisfacuta in intregime de sistemul de gestiune local de la o alta statie decat cea de unde ea a fost lansata. In acest caz cererea este transferata de sistemul distribuit nodului care detine datele pentru prelucrare (cerere la distanta);
- cererea poate fi satisfacuta numai preluand date de pe mai multe statii. In acest caz cererea este descompusa de catre sistemul distribuit in cereri locale si/sau cereri la distanta si transmise nodurilor corespunzatoare (cereri compuse), dupa care informatia de raspuns este agregata.



# Model entitate-asociere

## Model entitate asociere

- Aceasta sectiune prezinta conceptele de baza ale modelarii datelor utilizand modelul entitate asociere (model entityy-relationship ER), model conceptual de nivel inalt.

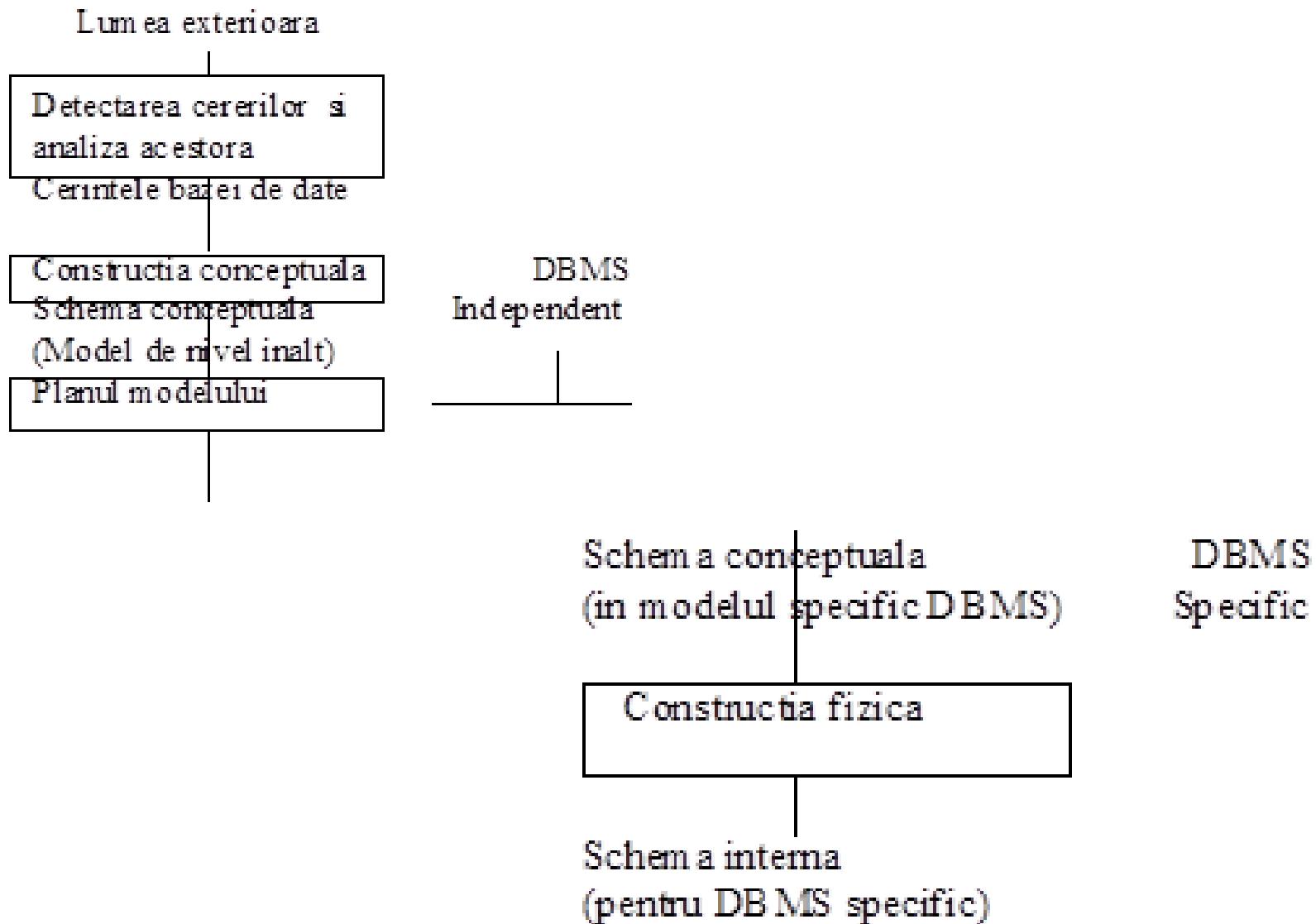
Caracteristici:

- Modelul asigura acoperirea nivelului extern privind perceperea si intelegerea de catre utilizatori a structurii datelor prin modelarea entitatilor si asocierilor.
- Nu se ocupe in nici un fel de detalii de stocare a datelor in memoria externa.
- A fost numit modelul datelor intrucit este compus dintr-un grup de concepte ce specifica structura bazei de date impreuna cu un set de operatii necesare manipularii lor.
- Modelul ER este o forma simpla de prezentare a structurii datelor, utilizat in procesul constructiei bazei de date, dar si in alte aplicatii in care apare entitati si relatii intre acestea.

## Etape ale modelarii:

- *Colectarea si analiza cerintelor*, discutii proiectant beneficiar.
  - **Scop**: stabilirea necesitatilor beneficiarului, a cerintelor referitoare la informatiile stocate, cat si a rezultatelor ce se doresc produse.
  - **Rezultat**: set de cerinte clar formulate.
- *Elaborarea schema conceptuala provizorie* utilizand un model de nivel inalt. Schema conceptuala reprezinta o descriere concisa a datelor utilizatorului incluzand descrierea detaliata a tipurilor de date, a relatiilor si restrictiilor pe care acestea trebuie sa le indeplineasca. Caracteristici:
  - Nu includ detalii de implementare.
  - Rezultatele pot fi comunicate beneficiarului chiar daca nu are cunostinte de baze de date.
  - Eliminarea eventualelor interpretari eronate a cerintelor de catre proiectant.

- ***Rafinarea schemei conceptuale*** in urma discutiilor cu beneficiarul.
- Elaborare ***schema conceptuala finala***.
- Constrictia ***schemei relationale***
- ***Constructia bazei de date*** (nivel conceptual) cu detaliile rezultate din modelul de nivel inalt. Caracteristici:
  - Independenta de DBMS
  - permita analiza complexitatii bazei de date si stabilirea mediului de baza de date utilizat pentru implementare.
- ***Transformarea modelului conceptual in model de implementare*** prin particularizarea nediului de baza de date utilizat.
- ***Constructia efectiva a bazei de date*** se specifica structura bazei ca model de stocare interna si sunt specificate constrangerile de integritate.



## Exemplu - model conceptual de nivel inalt

Cerintele pentru constructia unei baze de date aferenta unei companii, numita **COMPANIE**.

- Compania are un numar de angajati organizati pe departamente, si urmareste realizarea unor proiecte.
- Compania este organizata in departamente, fiecare departament are un nume, un numar de cod, un numar de angajati (minim 6), putind avea mai multe sedii.
- Un departament este implicat in mai multe proiecte, fiecare din ele are un nume, un numar de cod, si o locatie.
- Se pastreaza pentru fiecare angajat informatii de personal (nume, marca, cnp, etc).
- Fiecare angajat este afiliat la un departament, insa poate lucra la mai multe proiecte ce nu sunt neaparat coordonate de departamentul din care face parte. Trebuie stiut numarul de ore alocate lunar pentru fiecare proiect. De asemenea, fiecare angajat face parte dintr-un grup ce are un lider numit si supervisor.
- Angajatul poate avea persoane in intretinere (important la calculul impozitului).

## Entitati, atribute, relatii

- O **entitate** este un obiect real sau conceptual, cu o existenta independenta. O entitate poate fi un obiect fizic, o persoana particulara, automobil, companie, activitate, curs universitar, etc.
- Orice entitate are o serie de proprietati numite si **attribute** ce particularizeaza entitatea respectiva. De exemplu, pentru o entitate angajat se pot enumera o serie de atribute cum sunt nume, adresa, data nasterii, sex, salariu.

Ex:

- $e1=\{Nume=Vasile\ Ionescu, Adresa=Dreptatii - 211, Sector\ 6, Vîrstă = 40, Telefon=6621311\}$
- $e2=\{ Nume=U.P.B., Sediul=Splaiul\ Independentei - 313, Rector=V. Constantinescu\}$

## *Clasificare:*

- **Atribut compus.** Unele atribute pot fi impartite in parti mai mici cu semnificatie independenta. Un astfel de atribut se numeste si atribut *compus*. Un exemplu este cel al atributului adresa, ce poate fi subdivizat in attributele componente Oras, Judet, Cod postal, Strada.
- Atributele ce nu sunt compuse se numesc **attribute atomice**. Valoarea atributelor compusee se formeaza prin concatenarea valorilor atributelor atomice.
- Atribute de identificare – cu valoare diferita pentru fiecare entitate (utilizat la identificarea entitatii)
- Atribute de descriere – celelalte atribute

**Obs:** Multe atribute au valoare unica pentru o entitate particulara si sunt numite atribute cu o **singura valoare**. Spre exemplu vîrstă unei persoane. Există atribute ce pot lua mai multe valori dintr-un set finit, ca de exemplu gradele didactice într-o universitate, culoarea unui automobil, etc. Acestea sunt atribute cu **mai multe valori**. **Attributele derivate** sunt atributele ce se pot determina din relațiile ce există între ele ca de exemplu, vîrstă unei persoane ce se poate determina din data curentă și data de nastere.

- O baza de date este constituita in mod uzual din grupe de entitati similare caracterizate de aceleasi atribute. De ex. fiecare angajat este caracterizat de aceleasi atribute, dar fiecare membru al entitatii angajat va avea un set propriu de valori ce il characterizeaza. Entitatile ce sunt caracterizate de aceleasi atribute se numesc **entitati tip**. Pentru entitatea tip ANGAJAT avind atributele Nume, Vîrstă, Salariu, se da mai jos ca exemplu doi membrii:
  - e1(Popescu Vasile, 43, 3880)
  - e2(Dumitriu Petre, 32, 3240)
- Descrierea atributelor entitatii tip este numita **schema entitatii**
- Uzual, o entitate tip contine cel putin un atribut ce are valori distincte pentru fiecare instanta individuala (atribut de identificare). Aceste atribute se mai numesc si **chei candidate**. Dintre ele se alge un **atribut cheie**, iar valorile sale pot fi utilizate pentru identificarea in mod unic a entitatilor componente.
- O **cheie candidata** poate fi formata si din mai multe atribute (compusa) , avind proprietatea ca prin combinarea valorilor acestora se obtine un set de valori distinct.

- Spre exemplu, identificarea unui automobil nu poate fi facuta numai prin numarul de inmatriculare, ci si prin initialele judetului, deoarece exista in judete distincte automobile cu acelasi numar.
- O entitate poate avea mai multe atribute cheie, fiecare dintre acestea formand ceea ce numim o cheie candidata. Astfel, pentru studentii unei universitati avem cheile candidate: cod numeric personal, serie si numar buletin, numar matricol.
- Fiecare atribut al unei entitati tip are asociat un set de valori, numit si **domeniu**, ce specifica valorile posibile pe care atributul le poate lua.  
Ex:vîrstă unui angajat (16 - 70 ani), cnp (13 caractere), serie CI (2 caractere).. Matematic, atributul A al entitatii tip E poate fi definit ca o functie  $A : E \rightarrow P(V)$
- Valoarea atributului A pentru entitatea e, se va nota ca  $A(e)$ . Pentru un atribut simplu  $A(e)$  este o valoare cu un singur element, un atribut este null cand nu are valoare aplicabiula.

Pentru a se distinge, atributele compuse vor fi reprezentate intre paranteze. Se poate face prima descriere a modelului conceptual **COMPANIE**

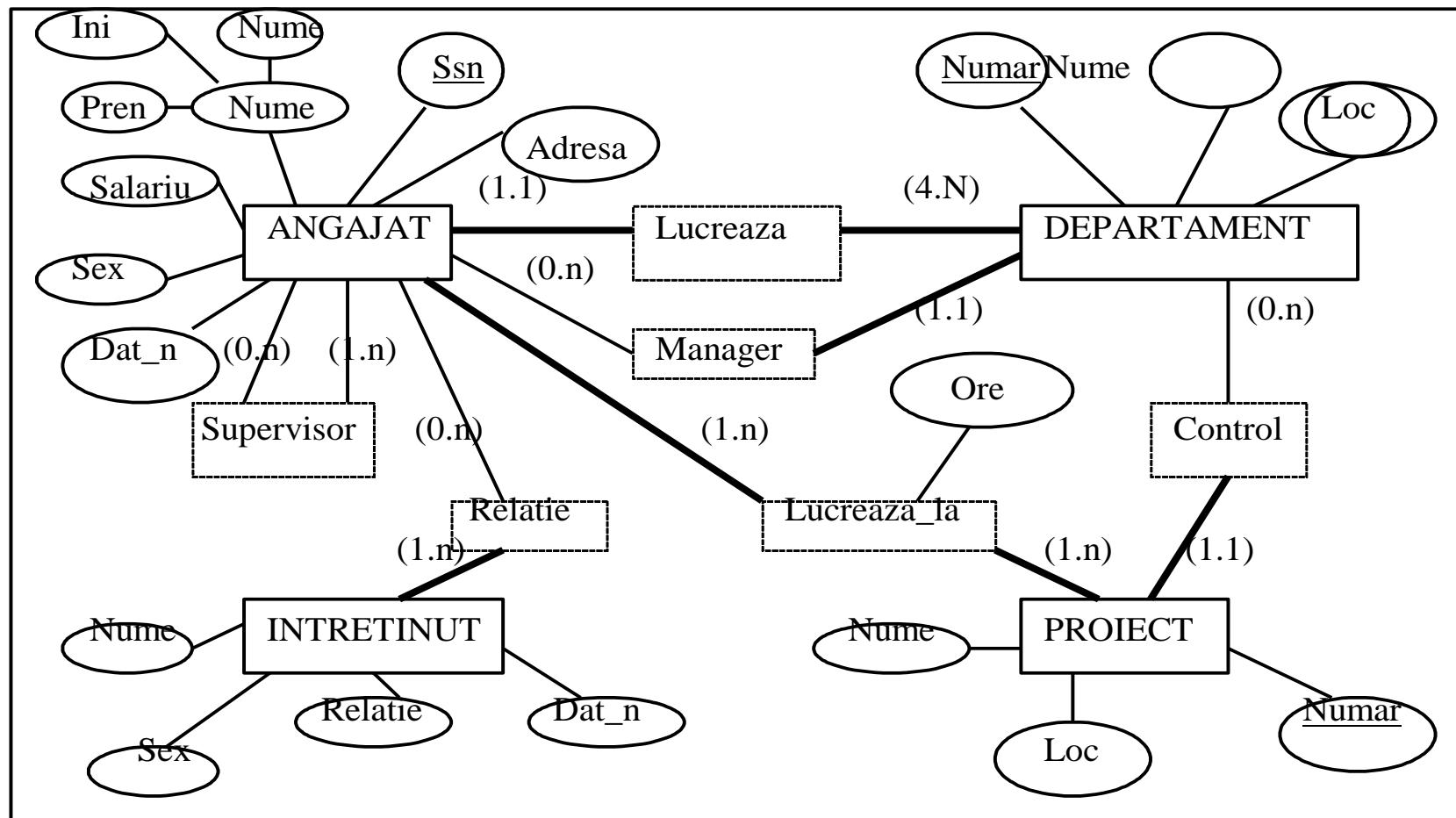
- O entitate tip **DEPARTAMENT** avand atributele Nume, Numar, Locatie (Sediu), Manager, Data de la care are aceasta functie.
- Entitatea tip **PROIECT** cu atributele Nume, Numar, Locatie, Departamentul ce coordoneaza proiectul, cu posibile atribut cheie oricare dintre Nume sau Numar.
- Entitatea tip **ANGAJAT** cu atributele Nume, SSN, Sex, Data nasterei, Adresa, Salariu, Departament si Supervisor. Atributele Nume si Adresa pot fi atribut compuse.
- O entitate tip **INTRETNUT** cu atributele Angajat, Nume, Sex, Data de nastere, Tip relatie (referitor la relatia dintre angajat si persoana aflata in intretinere).
- In aceasta incercare de definire nu s-a tinut cont de faptul ca un angajat poate sa lucreze la mai multe proiecte, nu apare in acest caz indicat numarul de ore alocate unui anumit proiect. Poate fi avut in vedere in cadrul entitatii ANGAJAT prin atributul compus Lucreaza\_la ce ar avea componente (Proiect,Ore). O alta alternativa poate fi introducerea unui atribut compus cu mai multe valori la entitatea PROIECT, atribut ce are componente (Angajat, Ore).

## Relatii tip (asocieri)

- O relatie tip (notata R) intre n entitati tip E1,E2,...,En este o asociere intre aceste entitati tip.
- O relatie instanta *ri* este un membru al relatiei tip si statueaza o asociere intre instante ale entitatilor tip. De exemplu relatia *LUCREAZA\_PENTRU* intre entitatile tip ANGAJAT si DEPARTAMENT asociaza fiecare angajat cu departamentul pentru care acesta lucreaza.
- Se defineste **rangul sau gradul unei relatii** tip ca numarul entitatilor tip participante la relatie.
- Daca relatia se stabeleste intre elementele aceleiasi entitati se spune ca s-a stabilit o **relatie recursiva**. Ca exemplu, relatia SUPERVIZOR se stabeleste intre angajati si supervisor, relatie in care ambele instante sunt membrii al aceleiasi entitati.
- **Restrictia de cardinalitate** este marcata de numarul relatiilor instant la care o entitate poate sa participe. Se simbolizeaza prin (m:n). Astfel exista relatii (0:1), (0:N), (1:1), (1:N), (M;N).
- **Restrictia de participare** specifica o conditie de existenta a unei relatii. Pot exista restrictii de participare **partiala sau totala**.

# Model E-R

## Modelul pentru BD Companie

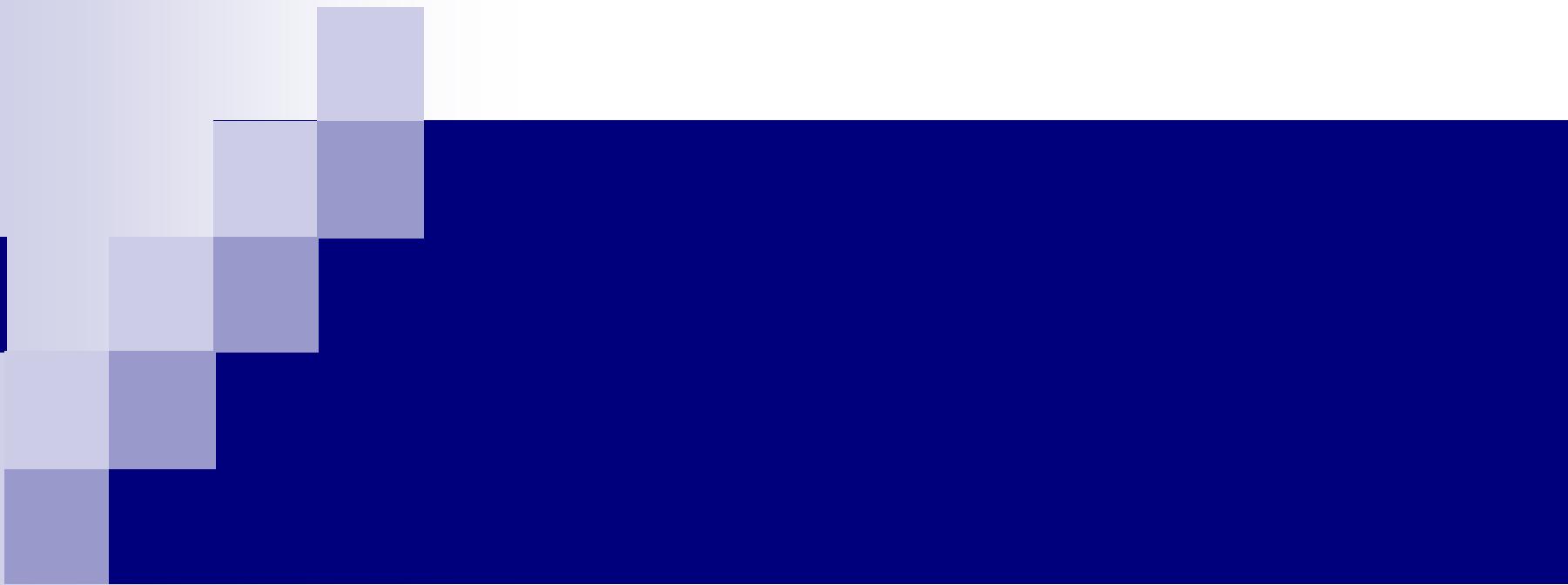


## Schema relationala conceptuala

- ANGAJAT (Nume, Ini, Pren, Ssn, Dat\_Na, Adr, Sex, Salariu, Sssn, D\_Nr, S\_ci, Nr\_ci, Marca)
- DEPARTAMENT(Dep\_Nume, Dep\_Nr, Dep\_location, Dep\_Manager, Dat\_Inc\_Manager)
- PROIECT(P\_Nume, P\_Nr, P\_Loc, Dep\_Nr)
- LUCREAZA\_LA(A\_Ssn, P-Nr, Ore)
- INTRETINUT(A\_Ssn , I\_Nume, Sex, Ssn\_i, Data\_N, Relatie)

## Obs:

1. S-a creat cate o relatie pentru fiecare entitate tip
2. Nu s-a decis inca ce se intampla cu atributele compuse
3. Cheile candidate sunt subliniate
4. La modelarea relatiei pentru fiecare entitate se modeleaza, daca este posibil si asocierile la care participa
5. Atentie la asocierea ANGAJAT – PROIECT
6. La rafinarea schemei conceptuale pot sa apara atribute noi (mai ales de descriere)
7. Atentie la participare si cardinalitate



# Memorarea datelor

# MEMORAREA DATELOR, FISIERE, ORGANIZARE

Mediul de stocare poate fi impartit in doua categorii:

- *stocarea primara* priveste acel mediu asupra caruia se poate opera direct de catre unitatea centrala (memoria interna).
- *stocarea secundara* se refera la toate mediile externe de memorare, cum sunt discul magnetic, banda magnetica, disc optic, caracterizate de capacitate mare, cost redus, dar cu un timp de acces mult mai mare decit memoria interna.

Justificarea stocarii externe a BD:

- volum prea mare pentru a fi stocate in memoria principala;
- datele trebuie sa ramana dupa intreruperea sursei de alimentare, adica au caracter permanent;
- costul pe unitatea de informatie este mult mai mic;
- poate fi asigurata transportabilitatea intre mai multe masini

# Disc magnetic (organizare)

- Informatia este stocata pe suprafata discului pe cercuri concentrice cu diametre distincte numite *piste* (*track*). La discurile compacte pistele de aceeasi raza formeaza un *cilindru*.
- Fiecare pista este impartita in mai multe *blocuri*, ce apartin unui *sector*. Marimea unui bloc este cuprinsa uzual intre 512 si 4096 octeti functie de sistemul de operare aferent. Separarea blocurilor se face in spatii de lungime fixa prin *gap interbloc*, ce contine informatii speciale de control scrise in momentul initializarii (formatarii) discului.
- Orice referire la informatiile stocate pe disc este o referire la un bloc. Adresa blocului este obtinuta prin combinarea indicativului suprafetei, indicativului pistei (intr-o suprafata) si indicativului blocului (in cadrul unei piste).
- Accesul la informatiile de pe disc se face prin citirea unui bloc si aducerea acestuia in memoria principala intr-o zona contigua numita si *buffer de intrare-iesire (I/O)*.
- Cand mai multe blocuri se transfera in memoria interna, existenta mai multor buffere face ca viteza de transfer sa creasca.

- Daca blocurile sunt contigue (**alocare contigua**) se consuma un timp de 15 - 60 ms pentru accesul la primul bloc, celelalte blocuri citindu-se in 1 -2 ms fiecare.
- Sistemul de calcul utilizat in baze de date are performante cu atit mai bune cu cat timpul de acces la informatia stocata pe disc este mai mic.
- Cand mai multe blocuri se transfera in memoria interna, existenta mai multor buffere face ca viteza de transfer sa creasca.
- In timpul citirii sau scrierii unui bloc unitatea centrala poate procesa alt bloc deja transferat in memoria interna.
- Prelucrarea paralela este posibila datorita existentei procesorului I/O pentru disc, ce poate lucra independent de unitatea centrala.

**Obs.** Din aceasta scurta descriere rezulta ca performantele oricarei aplicatii de baze de date sunt afectate de accesul la memoria externa.

**Concluzie:** Este foarte important ca numarul operatiilor de I/O pentru accesul la date sa fie cat mai redus.

## Alocarea blocurilor

- Alocare **contigua** (blocuri de pe acelasi cilindru sau cilindrtri vecini)
- Alocarea **cu legaturi** (pentru stocarea fisierelor secentiale in sisteme de operare) – **FAT**
- Alocare **indexata** (cu blocuri index pe diverse niveluri – structura arborescenta) metoda caracterizata de existenta unor pointeri de inlantuire a blocurilor
- Alocare bazata pe **HASH** Pointer = functie\_hash(camp\_hash) si indica grupul in care se gaseste inregistrarea. Utilizata atat la stocare cat si la citire

- Inregistrari cu lungime fixa


- Inregistrari cu lungime variabila


- În general, un bloc conține mai multe inregistrari, dar pot exista inregistrari ce incep într-un bloc și se termină în alt bloc, funcție de dimensiunea inregistrării și a blocurilor.

- Daca se noteaza numarul de octeti ai unui bloc cu B, si cu R numarul de octeti ai unei inregistrari (valabil doar pentru inregistrarile de lungime fixa), numarul inregistrarilor ce pot fi stocate intr-un bloc va fi dat de realtia:  
 $bfr = [B/R]$       in care,
- **bfr** - reprezinta *numarul de inregistrari pe bloc (blocking factor rate)*
- $[x]$  - reprezinta functia parte intreaga.
- In general R nu divide pe B, deci, daca nu sunt inregistrari ce trec dintr-un bloc in altul, in fiecare bloc folosit ramane un spatiu neocupat de  $B - (bfr * R)$  octeti.

Obs:

1. O tehnica uuala este definirea unor blocuri logice formate din mai multe blocuri fizice contigue.
2. Blocurile nu sunt complet ocupate

## Metode de cautare

1. **Cautare liniara (sequentiala)** specifica cautarii dupa chei de acces neordonate
2. **Cautare dihotomica.** (cautare prin segmentare – specifica cautarii dupa chei de acces ordonate)
- **Cautare binara.** Se considera un fisier sequential cu N inregistrari, ale carui chei sunt in ordine crescatoare  $k_1 < k_2 < \dots < k_N$ , cautarea avind ca scop gasirea inregistrarii cu cheia  $k$ .
- **Cautare binara uniforma**
- **Cautare Shar** (nod radacina  $\lceil \log_2 n \rceil$ )
- **Cautare Fibonacci** (nodurile sunt termeni ai sirului Fibonacci)

# Operatii la tabele de date

- **Find (Locate)** operatie de cautare a primei inregistrari ce satisface un criteriu de acces specificat.
- **Read (Get)** operatia de copiere din buffer intr-o variabila program sau zona de lucru pentru utilizator a unei inregistrari.
- **Find next** operatie de cautare a urmatoarei inregistrari ce satisface un criteriu de selectie specificat.
- **Delete** stergere a inregistrarii curente sau a grupului de inregistrari ce satisfac criteriul de selectie, si eventual rearanjarea tablei pe disc pentru a reflecta aceasta modificare..
- **Modify (Update)** operatie de modificare a valorilor campurilor din inregistrarea curenta sau din inregistrarile ce satisfac criteriul de selectie.

- **Insert** este operatia de adaugare a noi inregistrari in tabela de date, localizand blocul unde inregistrarile vor fi inserate, transfera blocul in bufferul intern (daca nu este aici), inscrie inregistrarea in buffer, scrie bufferul pe disc pentru a reflecta aceasta modificare. De foarte multe ori operatie de insert este inlocuita cu append, operatie mult mai simpla ce adauga o inregistrare la sfarsitul tableei.
- **Sort** operatie de ordonare fizica a tuturor inregistrarilor din tabela in ordinea specificata prin criteriul de selectie.
- **Index** este operatia de creare a unei ordini logice alta decat cea fizica. Operatia creaza structuri de date aditionale pe langa tabela de date, structuri ce definesc ordinea logica.

Trebuie facuta distinctia neta intre organizarea tableei si metoda de acces. Prin **organizare tablelei (table organization)** se intlege organizarea datelor in inregistrari, blocuri si structuri de acces. Prin **metoda de acces (acces metod)** se intlege grupul de programe ce permit aplicarea operatiilor de mai sus asupra tableei. In general metodele de acces sunt dependente de organizarea tableei

# Fisiere de date

Se cunosc doua moduri de organizare a informatiilor in fisier:

- organizare definita (fisier logic)
- organizare nedefinita (fisier fizic).. Accesul la un fisier cu organizare nedefinita are ca obiect obtinerea unor inregistrari fizice.

Principalele cerinte ale sistemelor de baze de date pot fi sintetizate astfel:

- stocarea unor cantitati mari de date;
- accesul usor la date;
- economia de spatiu la stocarea informatiilor;
- fiabilitatea si protectia datelor;
- capacitatea de reprezentare a structurilor lumii reale.

**Concluzie:** Alegerea metodei de organizare va determina performantele relative ale sistemului.

Pentru a judeca calitativ performantele se analizeaza de regula urmatoarelor aspecte:

- spatiul necesar pentru a pastra o inregistrare;
- timpul necesar pentru accesul la o inregistrare arbitrara;
- timpul necesar pentru accesul la urmatoarea inregistrare;
- timpul de stocare dupa inserarea unei noi inregistrari;
- timpul de reorganizare dupa adaugarea unei noi inregistrari;
- timpul de citire a intregului continut.

# Tehnica Hash

- O metoda de organizare primara ce asigura un acces rapid la inregistrari dupa criterii specificate este cea utilizind tehnica *hash*.
- Se foloseste si termenul de acces *direct*, indicand accesul relativ prin pozitia inregistrarii.
- Criteriul de cautare este dat de egalitatea conditiei la un singur camp numit si *camp hash*, care nu este obligatoriu un camp cheie, numit si *cheie hash*.
- Inregistrarile sunt organizate in grupuri ce ocupa unul sau mai multe blocuri. Functia hash aplicata cimpului hash produce un intreg ce indica adresa blocului in care inregistrarea dorita este stocata, daca aceasta exista.
- In functie de locul in care se aplica tehnica hash, se intalnesc o serie de variante numite: hash intern (internal hash) pentru fisiere temporare in memoria interna, hash extern (external hash) pentru fisiere stocate pe disc.

# Hash intern

Se aplica la stocarea si accesul la date din memoria interna pentru care se utilizeaza un spatiu aprioric alocat pentru disponerea inregistrarilor. Daca consideram marimea spatiului alocat pentru stocarea datelor in memoria interna este cuprins intre 0 si  $H-1$ , atunci spunem ca exista  $H$  sloturi. Vom alege o functie hash care transforma valoarea cimpului hash intr-un intreg cuprins intre 0 si  $H-1$ .

Nu exista o modalitate stricta privind alegerea functiei hash, insa dintre cele mai utilizate metode amintim:

- retinerea unui numar fixat de biti din valoarea campului hash;
- obtinerea restului impartirii valorii intregi a cimpului hash considerat ca o secventa de biti la  $H$ ;
- aplicarea unei functii injective pe anumite portiuni ale cimpului hash considerat ca sir de biti.

## Exemplu

Grup1
Grup2
Grup3
Grup n
Overflow 1
Overflow k

Grupuri initiale (date de valorile functiei hash)

---

Grupuri overflow

# Hash extern

- O tehnica hash similara se aplica si asupra datelor stocate pe disc, pentru a imbunatatii accesul la blocurile de date.
- Metoda hash aplicata la datele stocate pe suport extern va fi adaptata la caracteristicile mediului de stocare.
- Blocurile sunt grupate dupa o anumita strategie, un bloc sau un grup de blocuri formeaza un grup (cluster).
- Pentru accesul la disc in loc sa se specifica adresa inregistrarii la fiecare operatie se va specifica adresa clusterului sau adresa relativa in cluster.
- Functia hash indica un numar relativ al cluster-ului, ce poate fi utilizat la calculul adresei absolute.
- Adresa blocurilor disc poate fi pastrata intr un fisier header.
- La coliziuni o metoda de rezolvare este inspirata de la tehnica hash interna, prin inlantuirea de blocuri overflow.
- la fiecare cluster se adauga un pointer ce inlantuie inregistrarile cu aceeasi functie hash plasate intr-un bloc overflow.

# Tipuri de indecsi

- **Index primar.** Un index primar este realizat pe baza unui atribut cheie la o tabela ordonata dupa atributul (campul) cheie.
- **Indexul secundar** poate fi construit dupa un camp cheie al tablei de date, insa tabela nu este ordonata dupa acest camp.
- **Index de grup.** Daca campul de indecsare nu este un camp cheie, putand avea pentru mai multe inregistrari aceeasi valoare structura aditionala de acces poarta denumirea de index de grup (clustering index).

La o tabela de date se pot construi oricate structuri de index functie de atributele utilizate pentru constructia sa.

- **Index multinivel** atunci cand indexul se trateaza similar cu o tabela la care se construieste in continuare index. Exista mai multe metode de implementare pentru index multinivel prin arbori B sau arbori B+.

**Obs.** Trebuie inteleasa diferența majoră între indexare și sortare.

Indexarea este o metoda de acces logic pe cand sortarea se refera la reorganizarea tablei și produce la o alta ordine fizica.

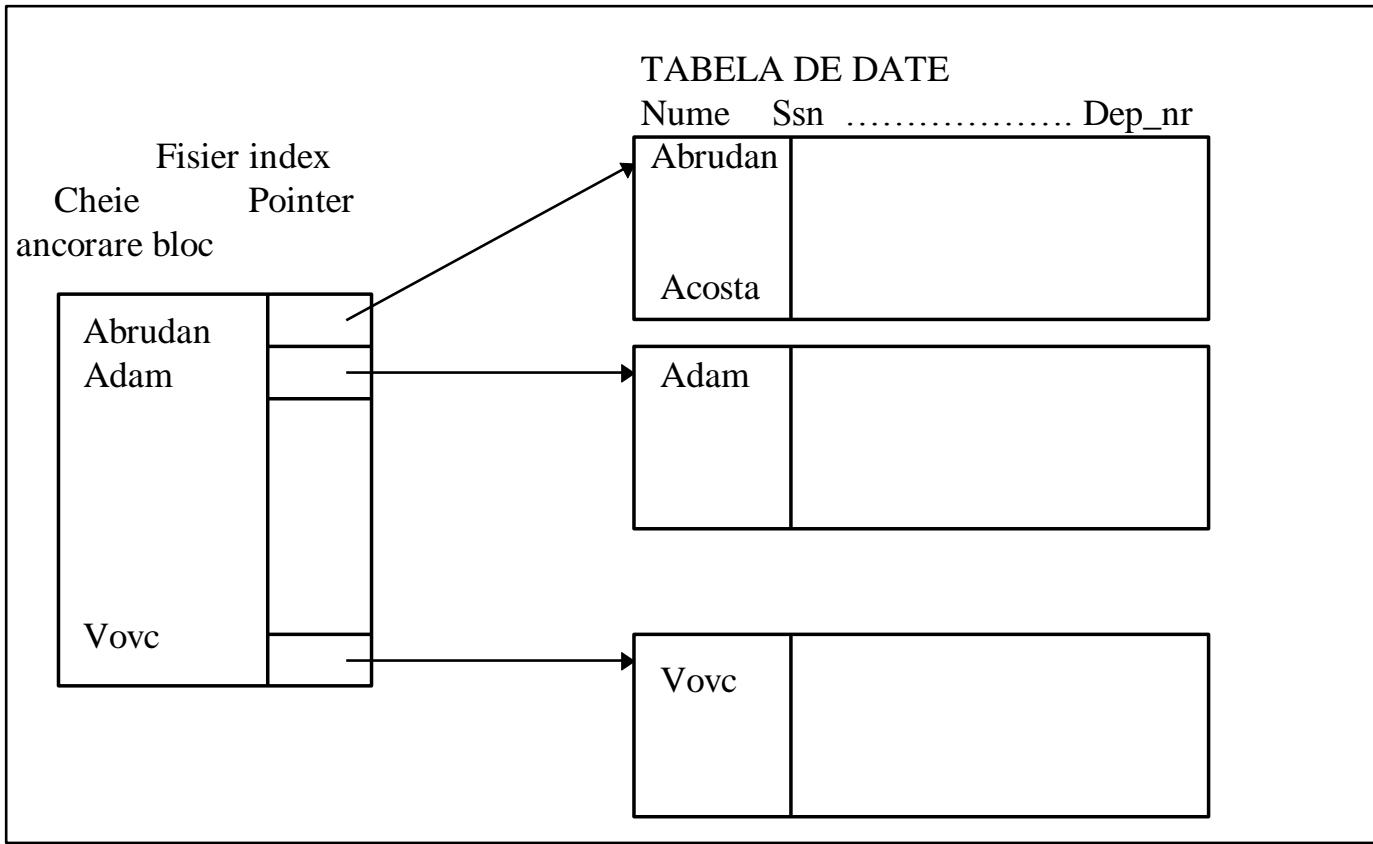
# Index primar

- Un index primar este un exemplu de index *nondens* datorita faptului ca are cate o intrare pentru fiecare bloc, fata de indexul *dens* ce asociaza cate o intrare pentru fiecare inregistrare a tabelei de date.

Indexul primar necesita mai putine blocuri pentru stocare datorita faptului ca;

- asociaza cate o singura inregistrare unui bloc de date;
- o intrare index ocupa mult mai putini octeti decat o inregistrare a tabelei de date. Ca efect cautarea intr-un index este mult mai rapida decat intr-o tabela de date, putind fi utilizate si metode de cautare binara deoarece indexul este totdeauna ordonat.
- O inregistrare pentru care valoarea cheii primare este  $k$  se gaseste in blocul  $i$  daca este indeplinita conditia  $k(i) \leq k < k(i+1)$ . Adresa blocului  $i$  este  $P(i)$  si inregistrarile in acest bloc sunt ordonate fizic dupa cimpul cheie.

# Exemplu de index primar



# Calcul performante (exemplu)

- Consideram o tabela de date ordonata dupa un camp cheie, ce contine  $r=30000$  inregistrari.
- Tabela este stocata pe disc in blocuri avand dimensiunea  $B=1024$  octeti.
- Toate inregistrarile sunt de lungime fixa si o inregistrare ocupa  $R=100$  octeti.
- Numarul de inregistrari pe bloc se calculeaza ca fiind  $bfr=[B/R]=[1024/100]$ , obtinand  $bfr=10$ . Nu este o pierdere faptul ca in fiecare bloc raman neutilizati 24 octeti, mult mai importante sunt performantele.
- Numarul de blocuri pentru tabela de date  $b=[r/bfr]=[30000/10]=3000$ , eventual plus 1.
- Cautarea binara in tabela de date necesita aproximativ  $\lceil \log_2 b \rceil + 1$  operatii de citire blocuri, deci  $\lceil \log_2 3000 \rceil + 1 = 13$  blocuri, numar de blocuri destul de mic rezultat din faptul ca tabela este ordonata.

- Cautarea binara in tabela de date necesita aproximativ  $\lceil \log_2 b \rceil + 1$  operatii de citire blocuri, deci  $\lceil \log_2 3000 \rceil + 1 = 13$  blocuri, numar de blocuri destul de mic rezultat din faptul ca tabela este ordonata.

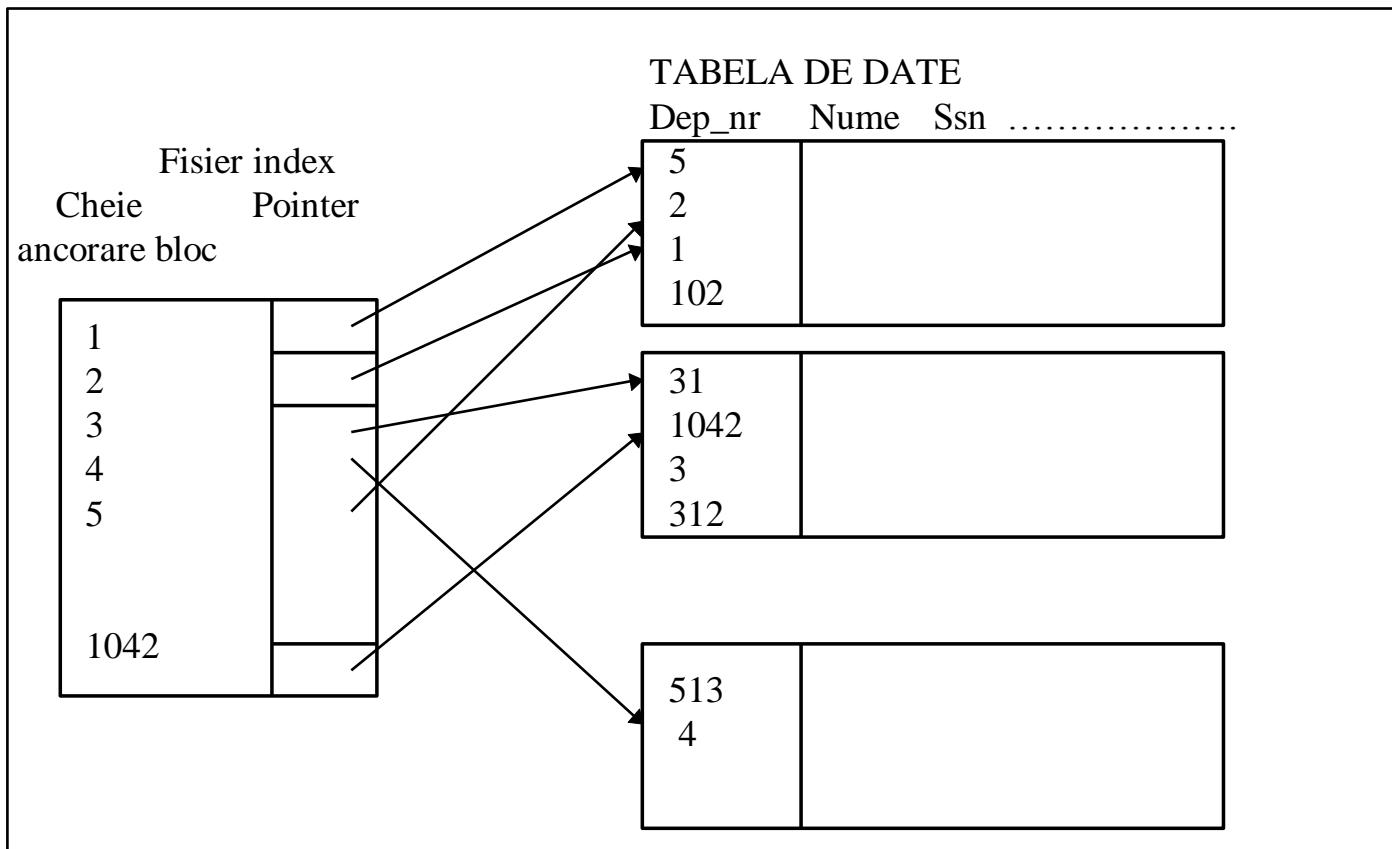
### Acces cu index

- Lungimea campului cheie de ordonare a tableei este de 9 octeti si ca un pointer catre un bloc poate fi stocat pe 6 octeti. O intrare index este  $R_i = V + P = 9 + 6 = 15$  octeti.
- Numarul intrarilor index pe bloc devine  $bfr_i = [B/R_i] = [1024/15] = 68$
- Numarul de inregistrari index este dat de numarul blocurilor tableei de date, in cazul de fata  $r_i = 3000$ . Numarul de blocuri index  $b_i = [r_i/bfr_i] + 1 = [3000/68] + 1 = 46$ .
- Cautarea binara in index necesita aproximativ  $\lceil \log_2 b_i \rceil + 1 + 1$  operatii de citire bloc, adica  $\lceil \log_2 45 \rceil = 5 + 1 + 1$ . In total pentru a se gasi inregistrarea va fi necesara citirea a 7 blocuri, 6 apartinand indexului si unul apartinand tableei de date.

# Index secundar

- Indexarea secundara se aplica la tabele neordonate dupa campul de indexare.
- *Indexul secundar* este o structura ordonata cu doua coloane, in care prima coloana este identica cu un camp al tabelei de date (camp de indexare), al doilea camp este un pointer catre blocul tabelei de date.
- Campul pentru care indexul este construit se numeste si *camp de indexare*. In concluzie, orice camp al unei tabele poate fi camp de indexare pentru indexul secundar.
- Un index secundar construit dupa un camp cheie din tabela de date. creaza o intrare index pentru fiecare inregistrare din tabela de date. Se obtine astfel un index dens, avand cate o intrare pentru fiecare inregistrare din tabela primara. Desigur ca, valorile din coloana pointer sunt neordonate, indicind de multe ori acelasi bloc intrucat intr-un bloc se pot gasi mai multe inregistrari.

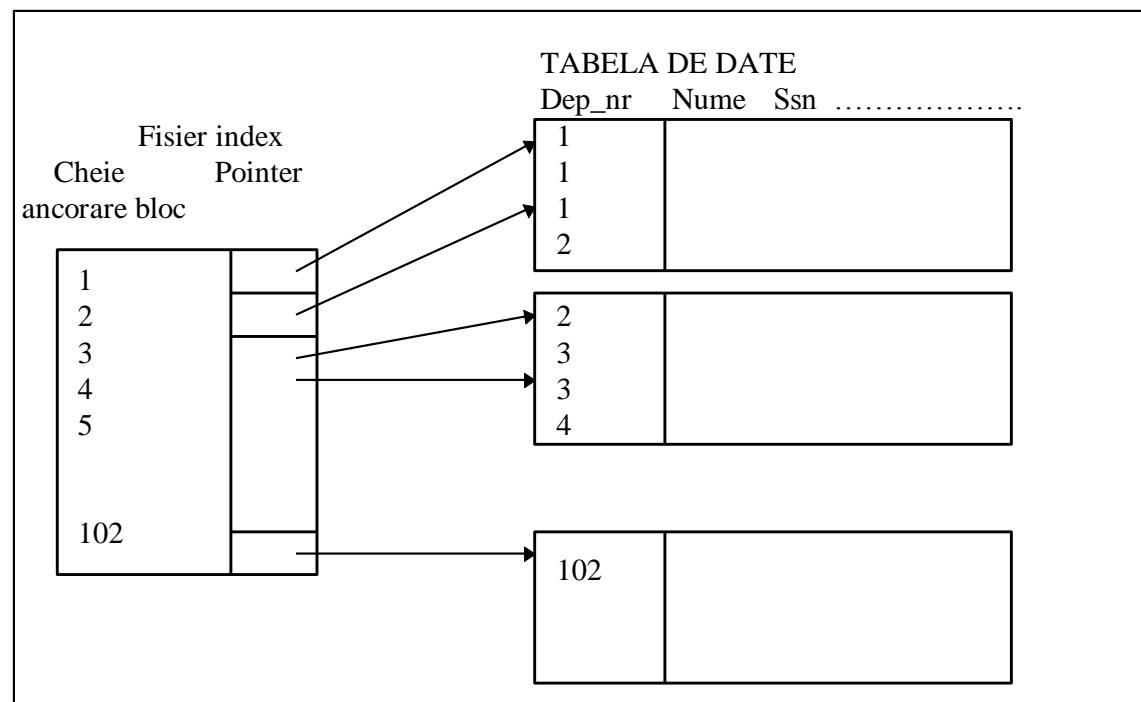
# Index secundar (tabela neordonata)



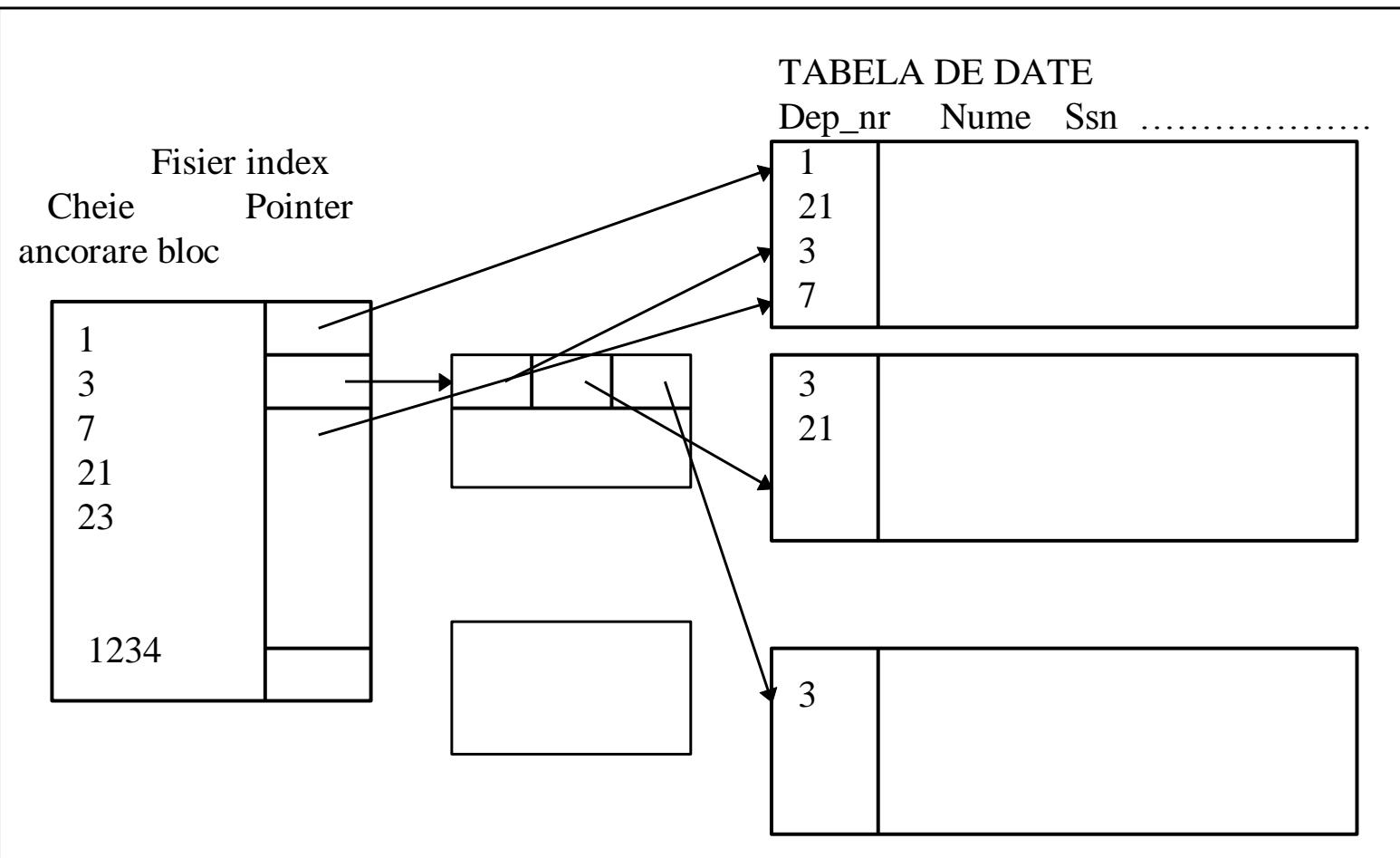
# Index de grup (cluster)

Astfel de indexi sint folositi cind inregistrarile tableei de date sint ordonate fizic dupa un camp noncheie, sau neordonate dupa un camp noncheie.

Tabela ordonata

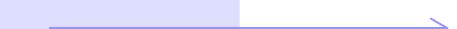


# Tabela nordonata

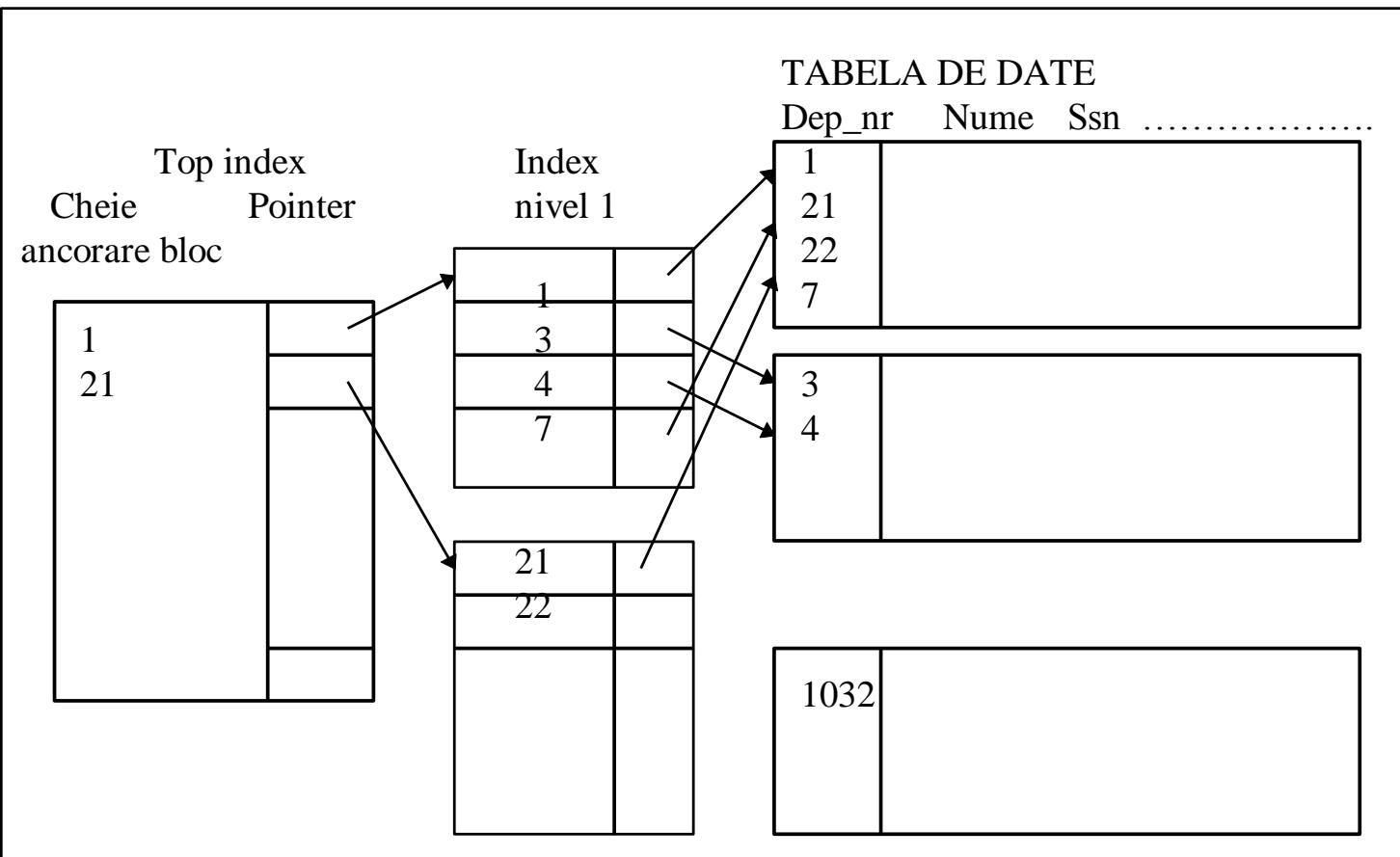


## Obs:

- Este mult mai rezonabil sa se pastreze intrari multiple in index decat inlantuirea cu blocuri de pointeri.

Camp indexare	Pointer
1	
2	
2	
2	
3	
3	

# Index multinivel (arbori B, arbori B+)



## Arbore B

Un arbore B de ordin p, construit dupa un cimp cheie al fisierului index satisface urmatoarele restrictii:

1. Fiecare nod intern in arborele B are structura

$\langle P_1, \langle K_1, P_{r1} \rangle, P_2, \langle K_2, P_{r2} \rangle, \dots, \langle K_{q-1}, P_{rq-1} \rangle, P_q \rangle$

cu  $q \leq p$ , in care,  $P_i$  este un pointer la un nou nod arbore numit si *pointer arbore*, pointerii  $P_{ri}$  fiind *pointeri data*, pointeri catre blocuri in tabela de date ce contin inregistrarea cu cheia  $k$ .

2 In fiecare nod  $K_1 < K_2 < \dots < K_{q-1}$

Valoarea inregistrarii cu cheia  $X$  in subarborele punctat de  $P_i$  este data de conditiile  $K_{i-1} < X < K_i$  pentru  $1 < i < q$ ,  $X < K_i$  pentru  $i=1$  si  $K_{i-1} < X$  pentru  $i=q$

4. Fiecare nod are cel mult p pointeri arbore

5. Fiecare nod exceptind radacina si nodurile frunza au cel putin  $[p/2]$  pointeri arbore. Nodul radacina are cel putin doi pointeri.

6. Un nod cu q pointeri arbore,  $q \leq p$ , contine  $q-1$  valori ale cimpului de indexare, deci  $q-1$  pointeri data

7. Toate nodurile frunza sunt la acelasi nivel, au aceeasi structura exceptind faptul ca toti pointeri arbore  $P_i$  sint nuli.

## Arbore B+

1. Fiecare nod intern are structura

$\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$

unde  $q \leq p$  si fiecare  $P_i$  reprezinta un pointer arbore.

2. In fiecare nod este indeplinita conditia

$K_1 < K_2 < \dots < K_{q-1}$ ,  $q \leq p$

3. Pentru toate valorile cimpului de cautare  $X$  in subarborele punctat prin  $P_i$  se indeplineste relatia

$K_{i-1} < X = K_i$  pentru  $1 < i < q$ ,  $X \leq K_i$  pentru  $i=1$  si  $K_{i-1} < X$  pentru  $i=q$

4. Fiecare nod intern are cel mult  $p$  pointeri arbore.

5. Fiecare nod intern exceptind radacina are cel putin  $[p/2]$  pointeri arbore. Nodul radacina are cel putin doi pointeri arbore daca este un nod intern si  $q \leq p$  pointeri data daca este nod frunza.

6. Un nod intern cu  $q$  pointeri,  $q \leq p$ , are  $q-1$  valori ale cimpului de cautare.

Pentru nodurile frunza sint indeplinite urmatoarele restrictii

1. Fiecare nod frunza este de forma

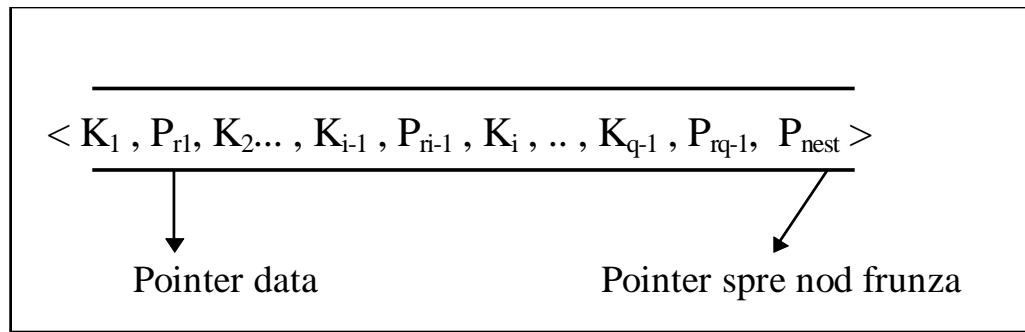
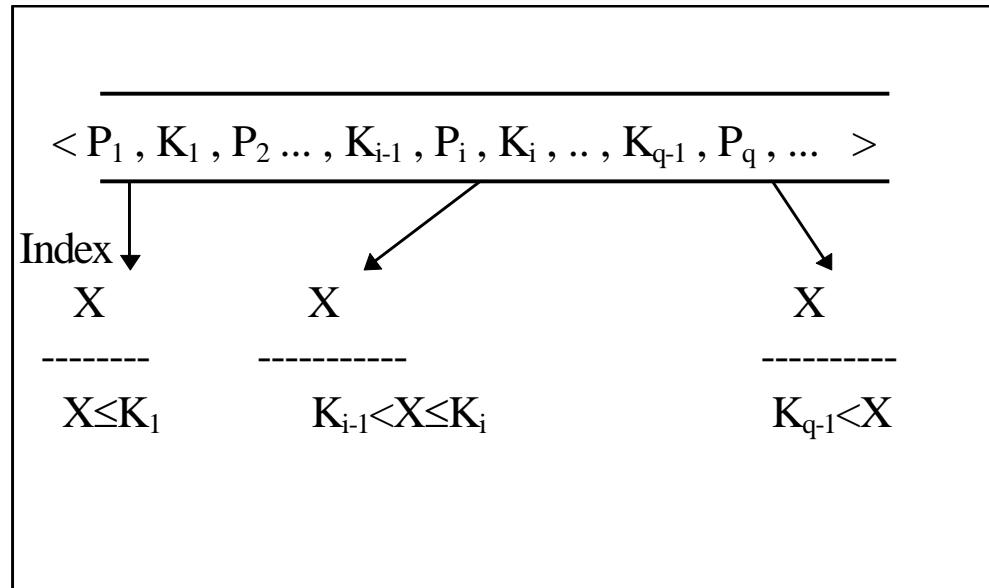
$\langle \langle K_1, P_{r1} \rangle, \langle K_2, P_{r2} \rangle, \dots, \langle K_{q-1}, P_{rq-1} \rangle, P_{next} \rangle$

unde  $q \leq p$ , fiecare  $P_{ri}$  este un pointer data si  $P_{next}$  indica urmatorul nod frunza al arborelui B+.

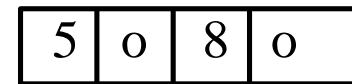
2. In fiecare nod frunza este indeplinita conditia

$K_1 < K_2 < \dots < K_{q-1}$ ,  $q \leq p$

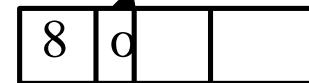
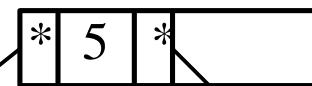
## Arbore B+



Arbore initial

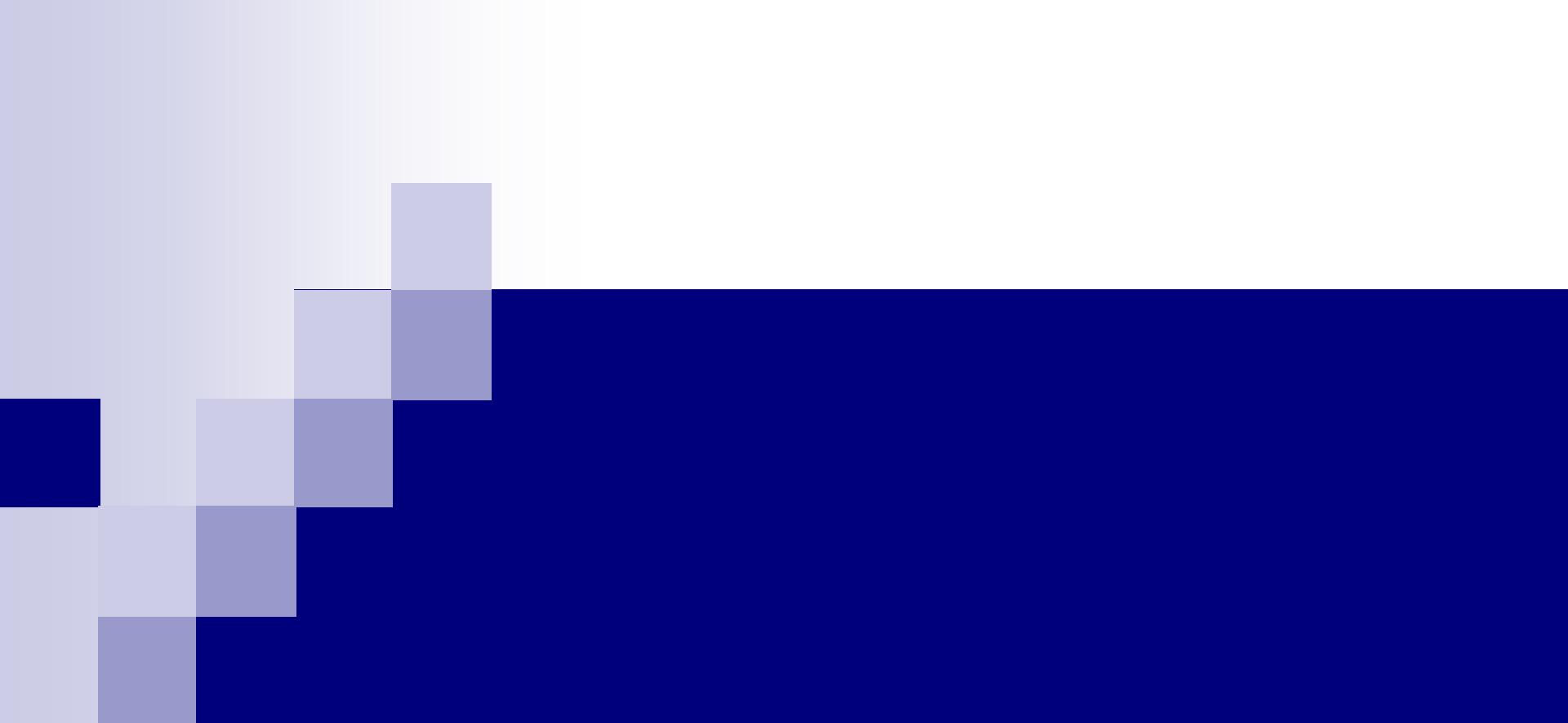


inserare 1



inserare 3 si 7





# Algebra relationala

# Modelul relational al datelor si algebra relationala

- In terminologia relationala, un rand al tableei este denumit *n-uplu*, o coloana poarta denumirea de *atribut* iar tabela se numeste *relatie*. Tipul datelor descriind tipul valorilor ce pot aparea in fiecare coloana este numit si *domeniu*. Un *domeniu* D este reprezentat de un set de valori atomice.
- Este foarte util sa se specifica pentru un domeniu un nume relevant pentru interpretarea corecta a valorilor datelor, ca de exemplu pentru relatia aferenta entitatilor student, numele relatiei poate fi asignat la valoarea *STUDENT*, iar atributurile sale cu semantica asociata ca in exemplul de mai jos:

NUME :Setul numelor persoanelor

NR\_TEL :Numarul de telefon reprezentat pe 10 digits

DEP\_NR :Numarul departamentului la care studentul studiaza

- Conceptul matematic ce acopera modelul relational este produsul cartezian al listei domeniilor. Un domeniu este un simplu set de valori ca de exemplu setul intregilor. Produsul cartezian al domeniilor  $D_1, D_2, \dots, D_n$  sau  $D_1 \times D_2 \times \dots \times D_n$  este setul tuturor  $n$ -uplurilor  $(v_1, v_2, \dots, v_n)$  in care  $v_i \in D_i$ . O relatie este reprezentata de orice subset al produsului cartezian al unuia sau mai multor domenii.

- O schema relatie  $R$ , data prin  $R(A_1, A_2, \dots, A_n)$  reprezinta un set al atributelor  $A_i$ . Fiecare atribut  $A_i$  joaca rolul numelui acelui domeniu  $D_i$  in relatia  $R$ .  $D_i$  este *domeniul* lui  $A_i$  si se noteaza  $\text{dom}(A_i)$ . O schema relatie este utilizata pentru descrierea relatiei  $R$ , unde  $R$  este *numele* relatiei. Se defineste *gradul sau ordinul relatiei* ca fiind numarul atributelor schemei relatiei. Fie de exemplu o schema relatie de grad 7, ce descrie studentii intr-o universitate, notat  $\text{ord}(R)$ :

STUDENT(Nume, SSN, Telefon, Adresa\_stabila, Adresa\_flotanta,  
Virsta, Directie\_specializare)

- O relatie  $r$ (sau *relatie instanta*) a unei relatii  $R(A_1, A_2, \dots, A_n)$  notata prin  $r(R)$ , se reprezinta ca un set al n-uplurilor  $t_i$ , asa ca  $r = (t_1, t_2, \dots, t_m)$ . Fiecare n-uplu t este o lista ordonata de n valori  $t = <v_1, v_2, \dots, v_n>$ , unde fiecare valoare  $v_i$  pentru  $i=1, n$  este un element al  $\text{dom}(A_i)$  sau o valoare speciala nula. Se utilizeaza de asemenea in mod curent termenii *relatie intensiva* pentru  $R$  respectiv *relatie extensiva* pentru relatie instanta  $r(R)$ . Numarul n-uplurilor unei relatii poarta debumirea de cardinalal relatiei,  $\text{card}(n)$ .
- Se poate da pentru o relatie urmatoarea definitie:

*O relatie  $r(R)$  este un subset al produsului cartezian al domeniilor ce defineste  $R$*

$$r(R) \subset (\text{dom}(A_1), \text{dom}(A_2), \dots, \text{dom}(A_n))$$

- Produsul cartezian contine toate combinatiile domeniilor. Deci, notand numarul valorilor sau *cardinalitatea domeniului D prin [D]*, si presupunind ca toate domeniile sunt finite, numarul total al n-uplurilor in produsul cartezian va fi dat de

$$[\text{dom}(A_1)]^* [\text{dom}(A_2)]^* \dots [\text{dom}(A_n)]^*$$

## Caracteristici ale relatiilor

- *ordinea n-uplurilor intr-o relatie.* Se spune despre doua relatii in care ordinea atributelor este diferita ca sunt identice.
- *ordonarea valorilor intr-un n-uplu.* Un n-uplu este o lista ordonata de n valori. Desi, la nivel logic ordinea atributelor si valorile lor nu este importanta, trebuie mentinuta corespondenta intre atribut si valoare. O relatie r a schemei relatiei  $R=(A_1, A_2, \dots, A_n)$  este un set finit de n-upluri  $r=(t_1, t_2, \dots, t_m)$  unde fiecare n-uplu  $t_i$  este o forma impachetata dupa structura R, cu D o reuniune a domeniilor atributelor,  $D=\text{dom}(A_1) \cup \text{dom}(A_2) \cup \dots \cup \text{dom}(A_n)$ . In aceasta definitie  $t(A_i)$  este o valoare din  $\text{dom}(A_i)$  pentru  $i=1, n$ .
- *valorile intr-un n-uplu.* Fiecare valoare in n-uplu este o valoare atomica ce nu poate fi divizata in fragmente. Daca sunt atribut care nu au valori aplicabile la anumite n-upluri, atunci va fi reprezentata o valoare speciala numita *valoare null*.
- *interpretarea unei relatii.* O schema relatie poate fi interpretata ca o declaratie sau un tip de asertiune.

- Se defineste *schema bazei de date relationale sau schema relationala* S ca un set al schemei relatiilor S=[R1,R2,...,Rm] impreuna cu un set al restrictiilor de integritate. Se defineste *instanta unei baze de date* relationale ca un set al relatiilor instanta [r1,r2,...,rm]. Se da mai jos, schema bazei de date relationale COMPANIE, avind relatiile ANGAJAT, DEPARTAMENT, PROIECT, LUCREAZA\_LA, INTRETINUT.
- ANGAJAT (Nume, Ini, Pren, Ssn, Dat\_Na, Adr, Sex, Salariu, Sssn, D\_Nr)
- DEPARTAMENT(Dep\_Nume, Dep\_Nr, Dep\_Manager, Dat\_Investire, Locatie, Telefon, Fax)
- PROIECT(P\_Nume, P\_Nr, P\_Loc, Dep\_Nr)
- LUCREAZA\_LA(A\_Ssn, P-Nr, Ore)
- INTRETINUT(A\_Ssn , C\_Nume, Sex, Data\_N, Relatie, Cnp\_i)

Vom defini un de operatii asupra relatiilor. Rezultatul este tot o relatie.

## Tipuri de operatii

- **Operatii unare.** Sunt operatii de ce aplica unei singure relatii. Din aceasta categorie fac parte:
  - Operatia **Select**
  - Operatia **Project**
- **Operatii binare**
  - Setul teoretic imprumutat din teoria multimilor
    - **Reuniune, Intersectie, Diferenta** (relatii compatibile)
    - Produs cartezian
  - Operatie specifica **JOIN**

## Operatia SELECT

- Relatia ce rezulta dupa operatia SELECT contine aceleasi atribute cu relatia initiala, asupra careia se face operatia. In general operatia SELECT are urmatoarea sintaxa:

$S_{<\text{conditie selectie}>}(<\text{nume relatie}>)$

Expresiile booleene specificate in  $<\text{conditie selectie}>$ , fac parte din clase de forma:

$<\text{nume atribut}> <\text{operatie comparatie}> <\text{valoare constanta}>$

sau

$<\text{nume atribut}> <\text{operatie comparatie}> <\text{nume atribut}>$

unde,

$<\text{nume atribut}>$  este numele unui atribut al relatiei definite;  $<\text{operatie comparatie}>$  este data de unul din operatorii  $\{=,<,<=,>,>=,\# \}$ , iar  $<\text{valoare constanta}>$  este o constanta.

Exemple:

- $S_{D\_NR=4}(\text{ANGAJAT})$
- $S_{\text{SALARIU}>4000}(\text{ANGAJAT})$

■  $S_{(D\_NR=3 \text{ .AND. } SALARIU>4000) \text{ .OR. } (D\_NR=4)}(\text{ANGAJAT})$

Semnificatia operatiilor booleene este:

- (cond1 .AND. cond2) adevarata daca ambele conditii sint adevarate, altfel falsa.
- (cond1 .OR. cond2) este adevarata daca fie conditia 1, fie conditia 2 este adevarata.
- (.NOT. cond) este adevarata atunci cand conditia este falsa.
- Faptul ca operatia se aplica la o singura relatie face ca operatorul de selectie sa fie *unar*. De asemenea, operatia se aplica la fiecare n\_uplu individual, iar gradul relatiei rezultante in urma unei operatii SELECT este egal cu gradul relatiei initiale R. In general:

**ord(REZ)=ord(R) si card(REZ)<=card(R)**

- Operatia SELECT este *comutativa* adica

■  $S_{\langle \text{cond1} \rangle}(S_{\langle \text{cond2} \rangle}(R)) = S_{\langle \text{cond2} \rangle}(S_{\langle \text{cond1} \rangle}(R))$

O cascada de operatii SELECT se pot combina utilizind operatorul AND.

$S_{\langle \text{cond1} \rangle}(S_{\langle \text{cond2} \rangle}(\dots(S_{\langle \text{condn} \rangle}(R))..)) = S_{\langle \text{cond1} \rangle}.\text{AND}.\langle \text{cond2} \rangle$   
 $\dots.\text{AND}.\dots.\text{AND}.\langle \text{condn} \rangle(R)$

# Operatia PROJECT

- Daca se considera o relatie similara cu o tabela atunci operatia SELECT selecteaza acele randuri din tabela ce corespund conditiei de selectie. Operatia PROJECT este utilizata pentru a selecta numai acele coloane ce corespund listei atributelor specificate. Se presupune ca se doreste obtinerea pentru angajatii companiei a numelui si prenumelui.

$P_{\text{NUME, PREN}}(\text{ANGAJAT})$

- Se observa ca forma generala a unei operatii PROJECT este  
 $P_{\langle \text{lista atributelor} \rangle}(\langle \text{nume relatie} \rangle)$

unde,

$P$  este simbolul utilizat pentru operatia PROJECT;

$\langle \text{lista atributelor} \rangle$  reprezinta lista atributelor relatiei specificate prin  $\langle \text{nume relatie} \rangle$ .

- Operatia PROJECT nu este in general comutativa. Aplicarea unei succesiuni de operatii PROJECT, daca succesiunea este valida, este echivalenta cu o singura operatie PROJECT.

$P_{\langle \text{lista1} \rangle}(P_{\langle \text{lista2} \rangle}(R)) = P_{\langle \text{lista1} \rangle}(R)$

- Operatia este valida numai daca lista2 contine toate atributele din lista1

Obs:  $\text{ord}(\text{REZ}) \leq \text{ord}(R)$ ,  $\text{card}(\text{REZ}) \leq \text{card}(R)$

## Combinatii SELECT – PROJECT

- Se considera ca exemplu, obtinerea relatiei ce contine numele, prenumele si salariul angajatilor ce lucreaza in departamentul 3. Pentru aceasta este necesara aplicarea atat a relatiei SELECT cat si a relatiei PROJECT. Aplicarea pe rand a operatiilor SELECT si PROJECT duce la crearea unei relatii intermediare notata in exemplul de mai jos REL1.

REL1 =  $S_{D\_NR=3}(\text{ANGAJAT})$

REZULTAT =  $P_{\text{NUME}, \text{PREN}, \text{SALARIU}}(\text{REL1})$

- A doua varianta, cea a utilizarii unei operatii compuse, asigura obtinerea relatiei rezultat fara a mai crea o relatie intermediara.

REZULTAT =  $P_{\text{NUME}, \text{PREN}, \text{SALARIU}}(S_{D\_NR=3}(\text{ANGAJAT}))$

- Pentru oricare dintre operatii atributele relatiei rezultat pot fi redenumite. Daca nu se face redenumire numele atributelor rezultatului se mosteneste de la relatia la care operatia se aplica
- Aplicarea secentelor de operatii poate rezolva si problema redenumirii atributelor in relatiiile intermediare. Astfel, redenumirea atributelor in relatia rezultat se exemplifica in urmatorul exemplu

TEMP =  $S_{D\_NR=5}(\text{ANGAJAT})$

REZ(P\_NUME, U\_NUME, SALARIU) =  $P_{\text{NUME}, \text{PREN}, \text{SALARIU}}(\text{TEMP})$

## Setul teoretic al operatiilor

- Prin setul teoretic se intelegh operatiile standard cum sunt REUNIUNE, INTERSECTIE, DIFERENTA. Operatiile din aceasta categorie se aplica asupra a doua relatii, deci fac parte din categoria operatiilor *binare*. Ca exemplu, se presupune ca se doreste lista numerelor de cod (Social Security Number) a tuturor angajatilor ce lucreaza in departamentul 3 sau sunt supervisori angajati ai departamentului 3. O modalitate de realizare este data mai jos:

$REL1 = S_{D\_NR=3}(\text{ANGAJAT})$

$REL2 = P_{SSN}(REL1)$

$REL3(Ssn) = P_{SSSN}(REL1)$

$REZ = REL2 \cup REL3$

- Aplicarea operatiei REUNIUNE asupra unor relatii nu este posibila intotdeauna. Referitor la operatiile de mai sus este necesar ca cele doua relatii sa aiba acelasi tip de n-upluri conditie numita si *compatibilitatea reuniunii*.

- Se spune ca doua relatii  $R(A_1, A_2, \dots, A_n)$  si  $S(A_1, A_2, \dots, A_n)$  sunt *compatibile reunii* daca au acelasi ordin, adica  $n=m$  si  $\text{dom}(A_i)$  este identic cu  $\text{dom}(B_i)$  pentru  $1 \leq i \leq n$ . Altfel spus, cele doua relatii au acelasi numar de atribute si atributele caracterizeaza acelasi domeniu.
- Fie doua relatii  $R$  si  $S$ . Pentru toate operatiile  $\text{ord}(REZ) = \text{ord}(R_1) = \text{ord}(R_2)$  Se pot defini operatiile teoretice pe cele doua relatii astfel:
- REUNIUNEA este operatia al carei rezultat include toate n-uplurile ce sunt fie in  $R$ , fie in  $S$ , fie in ambele relatii  $R$  si  $S$ . Executia elimina n-uplurile dupicate, relatia rezultanta se noteaza prin  $R \cup S$ .  
 $\text{Card}(REZ) \geq \max(\text{card}(R_1), \text{card}(R_2))$
- INTERSECTIA este operatia ce asociaza relatiei  $R \cap S$  acele n-upluri ce sunt continute atit in relatia  $R$  cit si in relatia  $S$ .  
 $\text{Card}(REZ) \leq \min(\text{card}(R_1), \text{card}(R_2))$
- DIFERENTA notata  $R \setminus S$  contine toate n-uplurile ce se gasesc in relatia  $R$  si nu se gasesc in relatia  $S$ .  $\text{Card}(REZ) \leq \text{card}(R_1)$

## Produs cartezian

- Rezultatul produsului cartezian intre relatiile  $R(A_1, A_2, \dots, A_n)$  si  $S(B_1, B_2, \dots, B_m)$  este o relatie  $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$  avind  $m+n$  atribute in ordinea specificata. In relatia rezultanta se combina un  $n$ -uplu al relatiei  $R$  cu fiecare  $n$ -uplu al relatiei  $S$ , asa ca, daca  $R$  are  $N$   $n$ -upluri si  $S$  are  $M$   $n$ -upluri, atunci  $Q=R \times S$  va avea  $M^N$   $n$ -upluri. Ca exemplu, se considera cerinta de obtinere pentru fiecare angajata de sex feminin lista numelor persoanelor pe care le are in intretinere. O posibilitate este:
  - $REL1 = S_{SEX='F'}(ANGAJAT)$
  - $REL2 = P_{NUME,PREN,SSN}(REL1)$
  - $REL3 = REL2 \times INTRETINUT$  (produs cartezian)
  - $REL4 = S_{SSN=A\_SSN}(REL3)$
  - $REZ = P_{NUME,PREN,C\_NUME}(REL4)$
  - $REL3$  este rezultatul aplicarii produsului cartezian intre  $REL2$  si relatia  $INTRETINUT$ .

**Obs:** Se constata ca operatia este expansiva, motiv pentru care multe SGBD-uri nu o implementeaza.

## Operatia JOIN

- Operatia JOIN este utilizata pentru combinarea n-uplurilor din doua relatii intr-o singura relatie, dar conditionat. Este una din operatiile specifice bazelor de date relationale cu mai mult de o relatie, permitind construirea de noi relatii pe baza celor existente. Se considera ca se doreste sa se obtina numele managerului fiecarui departament. Pentru aceasta este necesara combinarea n-uplurilor *ANGAJAT* cu n-uplurile *DEPARTAMENT* si selectarea acelora pentru care atributele Ssn si Dep\_manager au valori egale. Aceasta poate fi realizata prin utilizarea operatiei JOIN, astfel:

$REL1 = DEPARTAMENT ><_{DEP\_MANAGER=SSN} ANGAJAT$

$REZ = P_{DEP\_NUME, NUME, PREN}(REL1)$

- In concluzie operatia JOIN intre doua relatii R(A<sub>1</sub>,A<sub>2</sub>,..,A<sub>n</sub>) si S(B<sub>1</sub>,B<sub>2</sub>,..,B<sub>m</sub>) are structura

$R ><_{\text{cond join}} S$

avind ca rezultat o relatie Q cu n+m atribute Q(A<sub>1</sub>,A<sub>2</sub>,..,A<sub>n</sub>,B<sub>1</sub>,..B<sub>m</sub>) in aceasta ordine, fiind similara cu produsul cartezian dar, sunt retinute numai acele n-upluri ce satisfac in plus conditia de join.

- Structura tipica a unei conditii join este:  
 $<\text{conditie}> \text{ AND } <\text{conditie}> \text{ AND } <\text{conditie}> \text{ AND } \dots \text{ AND } <\text{conditie}>$   
unde, fiecare conditie este de forma  $A_i @ B_i$ , cu  $A_i$  atribut al relatiei R si  $B_i$  atribut al relatiei S. Sigur ca aceasta conditie nu poate fi adevarata decit daca atributele  $A_i$  si  $B_i$  reprezinta acelasi domeniu. Operatia notata simbolic @ esta asimilata cu una din operatiile  $\{=,<,<=,>=,>,\#\}$ . Se remarcă faptul ca n-uplurile la care atributele JOIN sunt nulle nu apar la rezultat.
- Cele mai obisnuite operatii JOIN invoca numai conditii de egalitate, motiv pentru care operatia se va numi EQUIJOIN. In aceasta situatie se obtin in rezultat mai multe perechi de atribut care au aceeasi valoare, ca urmare a indeplinirii egalitatii. Intrucit relatia obtinuta este superflua, o noua operatie numita si NATURAL JOIN elimina al doilea atribut din conditia de egalitate. Se va nota NATURAL JOIN cu \* avand semnificatia de EQUIJOIN urmat de eliminarea atributelor superflu. Se da mai jos un exemplu

PROIECT\_DEPART = PROIECT \*<sub>DEP\_NR=DEP\_NR</sub> DEPARTAMENT

- O operatie ceruta des in baze de date este DIVISION, operatie ce doreste sa dea raspuns la intrebari de tipul "*care este numele angajatilor ce lucreaza la aceleasi proiecte ca si angajatul Popescu Vasile ?*". Pentru rezolvare se procedeaza astfel:

1. se determina lista proiectelor la care lucreaza Popescu Vasile, obtinind o noua relatie REL2

$$REL1 = S_{\text{NUME}='POPESCU' \text{ .AND. } \text{PREN}='VASILE'}(\text{ANGAJAT})$$

$$REL2 = P_{P\_NR}(LUCREAZA\_LA * A\_SSN=SSN REL1)$$

2. se creeaza relatia ce include perechea atributelor  $\langle P\_NR, A\_SSN \rangle$ , pentru angajatii ce lucreaza la proiectul ce are numarul  $\langle P\_NR \rangle$

$$REL3 = P_{P\_NR,A\_SSN}(LUCREAZA-LA)$$

3. in final se aplica relatia DIVISION la doua relatii, ce va identifica angajatii doriti.

$$REL4(SSN) = REL3+REL2$$

$$REZ = P_{\text{NUME},\text{PREN}}(REL4 * \text{ANGAJAT})$$

## JOIN aditional

- Operatia JOIN se aplica doar la n-uplurile ce satisfac conditia de join. Ca exemplu, la NATURAL JOIN intre relatiile R si S, toate n-uplurile ce au atributie identice in cele doua relatii apar in rezultat, iar cele ce nu satisfac conditia sunt eliminate din rezultat. Un set de operatii numit si JOIN ADITIONAL este creat pentru pastrarea tuturor n-uplurilor din R , S sau din ambele. De exemplu, pentru lista tuturor angajatilor si lista departamentelor la care sunt manageri se poate aplica o operatie **JOIN ADITIONAL STING (LEFT OUTER JOIN)**, operatie cu simbolul  $]><$ , care va retine fiecare n-uplu din prima relatie sau relatie stanga R, iar acolo unde conditiile join nu sunt adevarate atributele lui S in rezultat au valori nule.

$\text{REL1} = (\text{ANGAJAT}) \text{ } ]><_{\text{SSN}=\text{DEP\_MANAGER}} (\text{DEPARTAMENT})$

$\text{REZ} = P_{\text{NUME}, \text{INI}, \text{PREN}, \text{DEP\_NUME}}(\text{REL1})$

- Similar pentru **JOIN ADITIONAL DREPT**
- Este posibil si **JOIN complet** in care se pastreaza si toate n-uplurile di dreapta si stanga ce nu indeplinesc conditia combinata cu n-upuri nule.

# Functii

## Functii agregat

- Primul tip de cereri ce nu pot fi exprimate prin algebra relationala sunt cererile specificind functii matematice aggregate asupra colectiilor de valori din relatie. De exemplu, se presupune cererea de calcul a sumei salariului tuturor angajatilor.
- Principalele functii ce pot fi aplicate valorilor numerice sunt SUM, AVERAGE, pentru tipuri de date ordonabile MAXIMUM, MINIMUM. Exista o functie ce se poate aplica indiferent de tipul de data, functia COUNT. Oricare din aceste functii se aplica la toate n-uplurile unei relatii.

Exemplu:

- Sa se obtina pentru intreaga companie numarul de angajati, media salariului, salariul maxim si salariul minim

Rez=  $F_{\langle \text{COUNT(ssn), AVERAGE(salariu), MIN(salariu), MAX(salariu)} \rangle}(\langle \text{Angajat} \rangle)$

**Obs:** Rezultatul este o relatie cu un singur n-uplu

## Functii de grup

- O a doua categorie de cereri se refera la aplicarea unor functii independente pentru fiecare grup unor grupuri de n-upluri. Ca exemplu, se considera grupul angajatilor care lucreaza in cadrul aceluiasi departament, particularizati prin valoarea atributului D\_nr, pentru care se poate determina suma salariului acestora.
- O operatie de tip functie de grup poate fi definita pentru specificarea acestor tipuri de cereri

$\langle\text{grupul atributelor}\rangle F_{\langle\text{lista functii}\rangle}(\langle\text{nume relatie}\rangle)$

unde,

- $\langle\text{grupul atributelor}\rangle$  este o lista a atributelor de grupare ale relatiei specificate prin  $\langle\text{nume relatie}\rangle$ ;
- $\langle\text{lista functii}\rangle$  se refera la o lista cu perechii  $\langle\text{functie(atribut)}\rangle$ .

## Exemplu:

Se considera baza de date *COMPANIE* la care se doreste sa se obtina pentru fiecare departament, numarul angajatilor si valoarea salariilor angajatilor departamentului. Relatia rezultata are attributele D\_NR, NUMAR\_ANGAJATI, MEDIE\_SALARIU obtinute prin aplicarea functiilor COUNT si AVERAGE in conjunctie cu atributul de grupare.

$$R(D\_NR, \text{NUMAR\_ANGAJATI}, \text{MEDIE\_SALARIU}) = \\ D\_NR F_{\text{COUNT}(\text{SSN}), \text{AVERAGE}(\text{SALARIU})}(\text{ANGAJAT})$$

**Obs.** Rezultatul este o relatie ce contine atatea n-upluri cate departamente distincte sunt in companie

- Daca in schimb nu se specifica atributele de grup atunci functia se aplica la toate n-uplurile relatiei, obtinind o relatie cu un singur n-uplu, De exemplu aplicarea operatiei

$F_{COUNT SSN, AVERAGE SALARIU}(ANGAJAT)$

## Alte exemple

Ex.1. Sa se gaseasca adresa tuturor angajatilor ce lucreaza pentru departamentul 'Cercetare'.

```
CERCET_DEPART = SDEP_NUME='Cercetare'(DEPARTAMENT)
CERCET_DEPART_ANAJAT = (CERCET_DEPART ><
D_NR=DEP_NR ANAJAT)
REZ = PNUME,INI,PREN,ADR(CERCET_DEPART_ANAJAT)
```

Ex.2. Pentru fiecare proiect localizat in 'Bucuresti', se cere lista numarului proiectului, numarul departamentului coodornator, numele managerului departamentului, adresa si data sa de nastere.

```
BUC_PROIECT = SP_LOC='BUCURESTI'(PROIECT)
COORD_DEP = (BUC_PROIECT >< DEP_NR=DEP_NR
DEPARTAMENT)
PROIECT_DEP_MANAGER = (COORD_DEP ><
DEP_MANAGER=SSN ANAJAT)
REZ =
PP_NR,DEP_NR,NUME,INI,PREN,ADR,DAT_NA(PROIECT_DEP_MANAGER)
```

*Ex.3.* Sa se gaseasca numele angajatilor care lucreaza la toate proiectele controlate de departamentul cu numarul 5.

$$\text{DEP5\_PROIECT}(P\_NR) = P_{P\_NR}(S_{\text{DEP\_NR}=5}(\text{PROIECT}))$$

$$\text{ANG\_PROIECT}(SSN, P\_NR) = P_{A\_SSN, P\_NR}(\text{LUCREAZA\_LA})$$

$$\text{REZ1} = \text{ANG\_PROIECT} + \text{DEP5\_PROIECT}$$

$$\text{REZ} = P_{\text{NUME}, \text{INI}, \text{PREN}}(\text{REZ1} * \text{ANGAJAT})$$

*Ex.4.* Sa se gaseasca lista tuturor angajatilor care au in intretinere doua sau mai multe persoane. In mod strict aceasta nu poate fi rezolvata prin operatii ale algebrei relationale. Pentru aceasta se utilizeaza functia COUNT ce nu face parte din operatiile algebrei relationale. In formularea de mai jos s-a presupus ca persoanele in intretinerea unui anumit angajat au nume distincte.

$$\text{REL1}(SSN, NR\_DE\_INTRETINUTI) = \underset{\text{COUNTC\_NUME}}{P_{A\_SSN}} F_{\text{INTRETINUT}}$$

$$\text{REL1} = S_{NR\_DE\_INTRETINUTI > 2}(\text{REL1})$$

$$\text{REZ} = P_{\text{NUME}, \text{INI}, \text{PREN}} (\text{REL1} * \text{ANGAJAT})$$

*Ex.5. Sa se gaseasca lista numelor angajatilor care nu au persoane in intretinere.*

$REL1 = P_{SSN}(\text{ANGAJAT})$

$\text{ANG\_CU\_INTRETINUT} = P_{A\_SSN}(\text{INTRETINUT})$

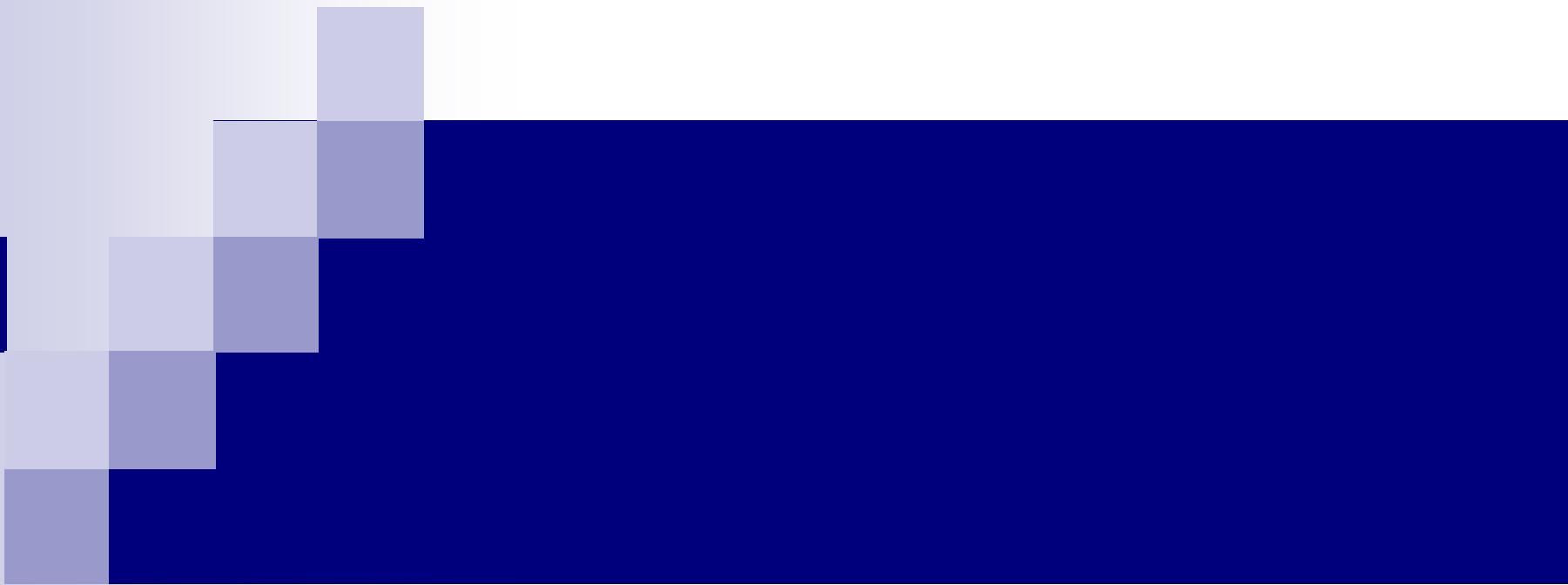
$\text{ANG\_FARA\_INTRETINUT} = (\text{REL1} - \text{ANG\_CU\_INTRETINUT})$

$\text{REZ} = P_{\text{NUME},\text{INI},\text{PREN}}(\text{ANG\_FARA\_INTRETINUT} * \text{ANGAJAT})$

*Ex.6. Sa se gaseasca lista managerilor care au cel putin o persoana in intretinere.*

$REL1 = \text{DEPARTAMENT} ><_{\text{Dep\_manager}=\text{A.ssn}} \text{INTRETINUT}$

$\text{REZ} = P_{\text{NUME},\text{INI},\text{PREN}}(\text{REL1})$



# SOL/DDL

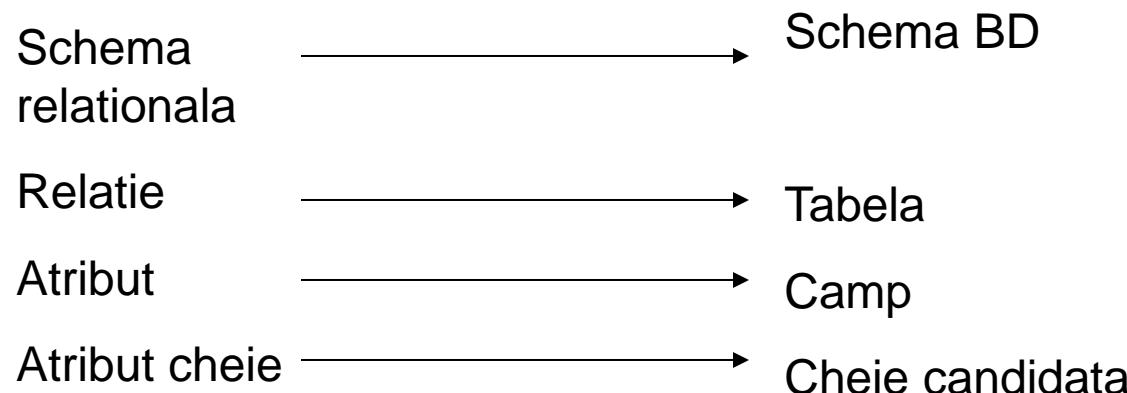
# Limbaje de cereri in modelul relational

- In cursul anterior s-au prezentat principalele operatii ale algebrei relationale.
- In acest curs se incepe prezentarea unui limbaj de implementare a acestor operatii intr-o baza de date relationala.
- In general aceste operatii sunt implementate intr-un limbaj de nivel inalt, operatiile se aplică la întreaga relație, cererea unei operatii a algebrei relationale este scrisă ca o secvență de operatii ce produc rezultatul dorit.
- In algebra relationala atunci cind se specifică o operatie complexa, utilizatorul are responsabilitatea ordinii de executie a operatiilor.
- Cele mai multe DBMS contin o interfata de nivel inalt, utilizatorul formulind cererea prin prisma rezultatului dorit urmând ca DBMS să ia decizia asupra modului de realizare.
- Cel mai cunoscut dintre limbajele implementate parțial sau complet în diferite DBMS este SQL (Structured Query Language)
- Initial SQL a fost numit și SEQUEL (Structured QUERy Language) construit și implementat de IBM și a fost destinat la implementarea interfeței pentru baza de date relatională numita și SYSTEM R.

- SQL este un limbaj de baze de date ce suporta declaratii pentru definirea datelor, modificarea si reorganizarea acestora. Deci SQL inglobeaza atat DDL, DML cat si VDL.
- In plus are facilitati privind definirea si vizualizarea bazei de date, crearea si intretinerea indexilor, definirea si intretinerea restrictiilor dintre tabele.
- Se descrie in continuare topica SQL implementata ca limbaj de interfata pentru diferite baze de date relationale, implementare ce acopera peste 90% din mediile de baze de date centralizate cunoscute.

## Model intern (model de stocare)

Avem urmatorul paralelism de interpretare intre algebra relationala si modelul de stocare



## Definitii constrangeri (integritate in BD):

- **Cheie candidata** – un camp sau o asociatie de campuri ce are valori distincte si NOT NULL pentru fiecare inregistrare din tabela unei baze de date
- **Cheie primara (PRIMARY KEY)** – o cheie candidata a tablei
- **Cheie straina (FOREIGN KEY)** – un camp (coloana) al unei table ce are asociat acelasi tip de data si este in asociere cu un camp al altrei table unde este o cheie primara sau unique
- **Camp UNIQUE** – un camp al unei table ce are valori distincte pentru fiecare inregistrare, dar care poate avea si valoare nula. Doua valori nule in acelasi camp al unei table sunt considerate distincte.
- **Camp NOT NULL** – un camp al unei table ce nu poate avea valori nule fara nici o restrictie de existenta a valorilor identice.
- **Camp CHECK** – constrangere prin care valorile permise indeplinesc o conditie de domeniu
- **Valoare DEFAULT** – un camp al unei table a carui valoare implicita este data de o expresie. La Insert cand valoarea nu este precizata, se atribuie automat acesta valoarea default

# Definirea datelor in SQL (SQL-DDL)

In acest grup se gasesc trei tipuri de instructiuni (CREATE, ALTER, DROP)

## Comanda CREATE TABLE

- Aceasta comanda este utilizata pentru definirea unui obiect de tip tabela prin care se specifica numele, coloanele, tipurile de data asociate acestora, formate si restrictii de integritate.
- Pentru fiecare camp se va specifica un nume, un tip de data si un domeniu de valori.
- Definirea structurii bazei de date COMPANIE fara constrangeri:
- **CREATE TABLE** Angajat (NUME VARCHAR(15),  
INI CHAR(1),  
PREN VARCHAR(12),  
SSN CHAR(13),  
DAT\_NA CHAR(8),  
ADR VARCHAR(30),  
SEX CHAR(1),  
SALARIU INTEGER,  
SSSN CHAR(13),  
D\_NR INTEGER);

```
CREATE TABLE departament
    DEP_NUME  VARCHAR(20),
    DEP_NR    INTEGER,
    DEP_MAN   CHAR(13),
    DAT_I_MAN CHAR(8));
CREATE TABLE proiect
    P_NUME   VARCHAR(15),
    P_NR     INTEGER,
    P_LOC    VARCHAR(30),
    DEP_NR   INTEGER);
CREATE TABLE lucreaza_la
    P_NR     INTEGER,
    ORE      DECIMAL(3,1));
CREATE TABLE intretinut
    C_NUME   VARCHAR(20),
    SEX      CHAR(1),
    DATA_N   CHAR(8),
    RELATIE  VARCHAR(10));
```

# Restrictii de integritate in definirea bazei de date

- O buna definire a unei baze de date trebuie sa surprinda relatiile intre tabele si restrictiile privind valorile posibile pentru anumite campuri.
- Acest obiectiv este atins prin specificarea **restrictiilor de integritate** impuse structurilor de date tabelare.

Tipuri de restrictii:

- PRIMARY KEY
- UNIQUE
- FOREIGN KEY
- NOT NULL
- CHECK

O restrictie poate fi precizata la nivel de camp, numita si in linie sau la nivel de tabela. De foarte multe ori o restrictie poate fi precizata fie la nivel de camp fie la nivel de tabela. Sunt totusi situatii in care o restrictie nu poate fi precizata la nivel de camp, fiind obligatorie precizarea sa la nivel de tabela.

## 1. Primary key

Restrictia precizeaza faptul ca un camp sau o asociatie de campuri reprezinta o cheie primara.

- Definire la camp in tabela angajat

```
CREATE TABLE Angajat (NUME VARCHAR(15),
                     INI CHAR(1),
                      PREN VARCHAR(12),
                      SSN CHAR(13) CONSTRAINT ang_PK
PRIMARY KEY,
                      DAT_NA CHAR(8),
                      ADR VARCHAR(30),
                      SEX CHAR(1),
                      SALARIU INTEGER,
                      SSSN CHAR(13),
                      D_NR INTEGER);
```

**Obs:** Portiunea **CONSTRAINT ang\_PK** este optionala

- Definire la nivel de tabela (lucreaza\_la)

```
CREATE TABLE lucreaza_la (A_SSN     CHAR(13),
                           P_NR      INTEGER,
                           ORE       DECIMAL(3,1),
                           CONSTRAINT lucr_PK PRIMARY KEY(A_SSN, P_NR));
```

OBS:

1. Daca cheia primara este formata dintr-un singur camp restrictia se poate formula fie la nivel de camp fie la nivel de tabela, pe cand in situatia in care cheia primara este o asociatie de campuri restrictia se precizeaza doar la nivel de tabela.
2. O tabela are o singura cheie primara
3. Cheia primara este una dintre cheile candidate
4. Se recomanda ca o tabela sa aiba o cheie primara
5. La definire se creaza automat o structura de acces rapid

## 2. FOREIGN KEY

Restrictia precizeaza faptul ca un camp este o cheie strina fiind cheie primara sau unique a altrei tabele

Definire in tabela angajat a cheii straine pentru campul D\_nr:

```
CREATE TABLE Angajat
  (
    NUME      VARCHAR(15),
    INI       CHAR(1),
    PREN      VARCHAR(12),
    SSN       CHAR(13) CONSTRAINT ang_PK
    PRIMARY KEY,
    DAT_NA    CHAR(8),
    ADR       VARCHAR(30),
    SEX       CHAR(1),
    SALARIU   INTEGER,
    SSSN      CHAR(13),
    D_NR      INTEGER CONSTRAINT ang_FK
  )
```

REFERENCES departament(D\_nr) FOREIGN KEY(D\_nr) on delete cascade|on delete set NULL|on delete set DEFAULT|no action);

## OBS:

1. O cheie strana face referire la o tabela in care campul referit este o cheie primara sau unique
  2. Modul de tratare a integritatii este definit prin specificarea actiunilor intreprinde de SGBD.
- Pentru introducerea unei inregistrari noi cu o valoare specificata a cheii straine este necesar ca in tabela referita aceasta valoare sa existe
  - La stergerea inregistrarii din tabela de referinta cu valoarea specificata in tabela cu restrictia de cheie strana se produce:
    1. Stergerea inregistrarilor daca s-a precizat **on delete cascade**
    2. Asignarea valorii NULL daca s-a precizat **on delete set NULL**
    3. Asignarea valorii **DEFAULT** daca s-a precizat o valoare implicita
    4. Nepermisa stergerea cu **NO ACTION** atata timp cat exista o inregistrare referita

**Obs:** Valorile NULL sunt considerate distincte

### 3. UNIQUE

Restrictie impusa oricarui camp prin care nu se accepta valori identice diferite de NULL

Exemplu in tabela departament

```
CREATE TABLE departament (DEP_NUME VARCHAR(20),
                          DEP_NR    INTEGER CONSTRAINT dep_PK
                                      PRIMARY KEY,
                          DEP_MAN   CHAR(13) CONSTRAINT dep_FK
                                      REFERENCES angajat(SSN) FOREIGN KEY on delete cascade,
                          DAT_I_MAN CHAR(8) UNIQUE);
```

Sau

```
DAT_I_MAN CHAR(8) CONSTRAINT data_unic UNIQUE
```

Sau la tabela

```
CONSTRAINT data_unic UNIQUE(DAT_I_MAN)
```

Pot exista mai multe campuri avand restrictia UNIQUE

## 4. NOT NULL

Restrictia statueaza faptul ca valoare campului nu poate fi nula la nici o inregistrare

Fie tabela angajat

CREATE TABLE Angajat

INI	CHAR(1)
PREN	VARCHAR(12) NOT NULL,
SSN	CHAR(13) CONSTRAINT ang_PK
PRIMARY KEY,	
DAT_NA	CHAR(8) NOT NULL,
ADR	VARCHAR(30),
SEX	CHAR(1),
SALARIU	INTEGER,
SSSN	CHAR(13),
D_NR	INTEGER CONSTRAINT ang_FK

REFERENCES departament(D\_nr) FOREIGN KEY on delete cascade|on delete set NULL| no action|on delete default);

**Atentie:** Si SSSN este o cheie strana ce refera tot tabela Angajat

## 5. CHECK

Restrictie privind valorile posibile ale unui camp ce poate fi atasata unui camp sau global mai multor campuri

Ex: Sa consideram tabela Student intr-o universitate la care pentru campurile “an studiu” si bursa se impun restrictii de valori intre 1 si 4, respectiv trei valori distincte posibile pentru bursa 120, 160 210.

CREATE TABLE Student (nume VARCHAR(10) NOT NULL,...

“an studiu”      INTEGER CHECK(“an studiu” BETWEEN 1  
and 5),  
bursa              INTEGER CHECK(bursa = 120 or bursa = 160  
or bursa = 210).....)

sau

CREATE TABLE Student (nume VARCHAR(10) NOT NULL,...

“an studiu”      INTEGER,  
bursa              INTEGER,

CONSTRAINT bursa\_an CHECK(“an studiu” >0 and “an studiu”<5  
and (bursa = 120 or bursa = 160 or bursa = 210)).....)

## OBS:

- In primul exemplu au fost puse restrictii la nivel de campuri pe cand in al doilea restrictiile sunt puse global la nivel de tabela
- Campul an studiu avand in denumire caracterul blank a fost inclus intre semne de grupare “ ”
- S-a utilizat atat o conditie logica cat si functia between
- Au fost specificate si alte restrictii in structura tablei, cum este restrictia NOT NULL.

Valoare **DEFAULT** (nu este constrangere)

De multe ori se specifica la un camp si valoarea sa implicita.

CREATE TABLE Student (nume VARCHAR(10) NOT  
NULL,...

“an studiu” INTEGER **DEFAULT 1**,  
bursa INTEGER,

CONSTRAINTS CHECK(“an studiu” >0 and “an studiu” <5  
and (bursa = 120 or bursa = 160 or bursa = 210)).....)

## Stergere si modificare structura table

1. Drop table comanda pentru stergerea unei table (stergere obiect)

**DROP TABLE Angajat**

2. Alter table modificarea structurii unei table

**ALTER TABLE nume\_tabela TIP\_modif (definitie)**

- Adaugarea unui camp

**ALTER TABLE angajat ADD (nume\_camp tip\_data [DEFAULT expresie] [constrangere])**

Ex:

**ALTER TABLE angajat ADD (studii VARCHAR(15) DEFAULT “medii”  
CONSTRAINT ang\_stud NOT NULL)**

Posibil CHECK pentru studii “gimnaziale”, “medii”, “superioare”,  
“doctorat”

OBS:

- Daca tabela contine cel putin o inregistrare noua coloana va avea valori nule
- Impunerea unei restrictii NOT NULL se poate face doar daca tabela nu are nici o inregistrare
- Coloana adaugata este ultima coloana in ordine fizica

- Stergerea unui camp

```
ALTER TABLE nume_tabela DROP COLUMN nume_coloana  
[CASCADE CONSTRAINTS]
```

Sau

```
ALTER TABLE nume_tabela DROP (lisata coloane) [CASCADE  
CONSTRAINTS]
```

Ex:

```
ALTER TABLE angajat DROP (SSSN)
```

```
ALTER TABLE angajat DROP COLUMN SSSN
```

OBS:

- Cererea se poate executa atat la tabele cu inregistrari cat si la tabele fara inregistrari
- Daca tabela contine o singura coloana aceasta nu poate fi stearsa
- Optiunea **CASCADE CONSTRAINTS** sterge suplimentar toate restrictiile de integritate in care sunt implicate coloanele sterse inclusiv cele de tip FOREIGN KEY

- Stergerea este o operatie costisitoare (din perspectiva resurse) si Oracle propune o operatie in doua etape:

## 1. Marcare ca neutilizata cu SET UNUSED

Ex:      **ALTER TABLE nume\_tabela SET UNUSED (lista coloane)**

Sau

**ALTER TABLE nume\_tabela SET UNUSED COLUMN nume\_coloana**

Marcarea ca neutilizate are efectul:

- Nu mai apar in structura afisata la DESCRIBE
- Nu mai pot fi folosite in cereri SQL
- Se pot adauga coloane cu acelasi nume cu cel al coloanelor sterse

## 2. Stergerea efectiva a coloanelor marcate ca neutilizate prin

**ALTER TABLE nume\_tabela DROP UNUSED COLUMNS;**

- Modificarea definirii unei coloane

`ALTER TABLE nume_tabela MODIFY (nume_coloana [tip_data]  
[DEFAULT expresie] [constrangere])`

Ex: `ALTER TABLE angajat MODIFY (D_nr INTEGER CONSTRAINT  
ang_FK REFERENCES departament(D_nr) FOREIGN KEY on  
delete set NULL)`

Efecte:

- Poate fi schimbat tipul de date al coloanei
- Poate fi asociata o noua valoare implicita
- Se poate adauga o constrangere
- Pot fi realizate toate in aceeasi operatie

OBS finala: Pentru ca operatiile DDL sa fie permise utilizatorul trebuie sa detina drepturile necesare setate prin sistemul de administrare.

- Adaugarea unei constrangeri

ALTER TABLE nume\_tabela ADD [CONSTRAINT nume\_constrangere]  
tip(coloana)

Ex: In tabela departament campul d\_nume nu poate fi NULL sau sa  
aiba valori identice si lungimea minima a sa este de 7 caractere:

ALTER TABLE departament ADD CONSTRAINT nume\_nenul CHECK  
(d\_nume IS NOT NULL)

ALTER TABLE departament ADD CONSTRAINT nume\_unic  
UNIQUE(d-nume)

ALTER TABLE departament ADD CONSTRAINT nume\_7 CHECK  
(LENGTH(d\_nume)>6)

- Stergerea unei constrangeri

ALTER TABLE nume\_tabela DROP PRIMARY KEY [CASCADE]

ALTER TABLE nume\_tabela DROP UNIQUE (lisata\_coloane)  
[CASCADE]

ALTER TABLE nume\_tabela DROP CONSTRAINT nume [CASCADE]

Efect:

- DROP PRIMARY KEY sterge constrangere de tip cheie primara chiar daca constrangerea nu are un nume asociat.
- DROP CONSTRAINT specifica stergerea unei constrangeri cu nume asociat
- Optiunea CASCADE se aplica daca sunt constrangeri dependente si se specifica stergerea acestora.

- Activarea si dezactivarea unei constrangeri

## 1. Dezactivare

`ALTER TABLE nume_tabela DISABLE PRIMARY KEY [CASCADE]`

`ALTER TABLE nume_tabela DISABLE UNIQUE (lisata_coloane)  
[CASCADE]`

`ALTER TABLE nume_tabela DISABLE CONSTRAINT nume  
[CASCADE]`

## 2. Activare

`ALTER TABLE nume_tabela ENABLE PRIMARY KEY`

`ALTER TABLE nume_tabela ENABLE UNIQUE (lisata_coloane)`

`ALTER TABLE nume_tabela ENABLE CONSTRAINT nume`

Efecte:

- Optiunea CASCADE duce la dezactivarea suplimentara a tuturor constrangerilor dependente
- La ENABLE CONSTRAINT se verifica consistenta datelor si daca constrangerile nu sunt conforme cu datele nu activeaza constrangerea si transmite mesaj de eroare.

- Golirea unei tabele (pare mai degraba o instructiune DML)

**TRUNCATE TABLE nume\_tabela [REUSE STORAGE]**

Ex: golirea tabelei intretinut

**TRUNCATE TABLE intretinut**

Optiunea **REUSE STORAGE** specifica faptul ca spatiul ocupat de liniile sterse ramane alocat tabelei si poate fi utilizat pentru inserari ulterioare

- Redenumirea unei tabele

**RENAME nume\_vechi TO nume\_nou**

Ex: **RENAME departament TO dep**

- Comentarii. Tabelele si coloanele pot avea asociat comentariu

**COMMENT ON TABLE nume\_tabela IS 'text asociat'**

**COMMENT ON COLUMN nume\_tabela.nume\_coloana IS 'text'**

Comentariile se tin in **USER\_TA\_COMENTS**, respectiv in

**USER\_COL\_COMMENTS** si **ALL\_COL\_COMMENTS**

# Interogarea bazelor de date

Cererile SQL fac parte din componenta DML. O fraza SQL include mai multe clauze ce sunt execute dupa optimizare de catre SQL.

Structura generala:

**SELECT lista campuri si functii**

**FROM lista table**

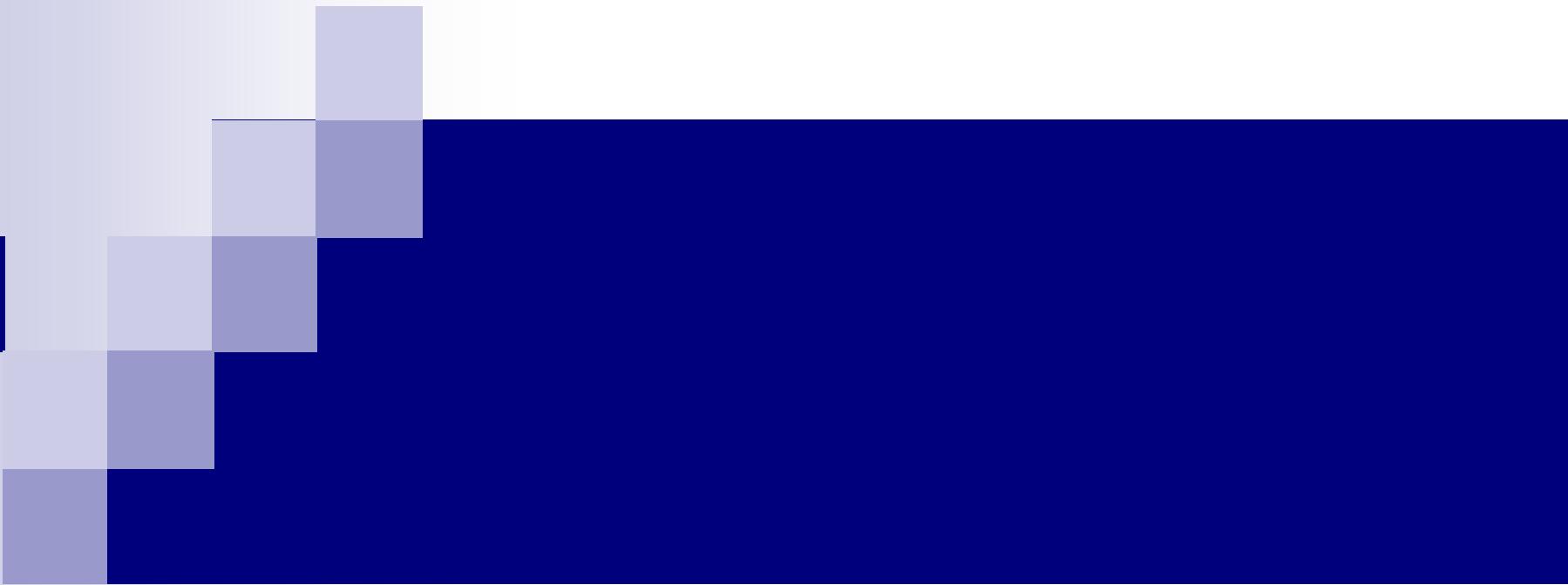
**WHERE conditii de tip select, join**

**GROUP BY lista campuri grupare**

**HAVING conditii pe functii**

**ORDER BY lista campuri grupare**

- Numai clauzele SELECT, FROM sunt obligatorii celelalte sunt dependente de natura cererii. Ordinea clauzelor group by, having, order by poate fi modificata
- Rezultatul executiei este tot o tabela, de regula temporara si se suprainscrie la executia oricarei interogari.



# SQL/DML

# Dictionarul de date

- Un SGBD relational necesita ca datele interne sa fie organizate si stocate in acelasi mod cu datele utilizatorilor si aplicatiile ce le folosesc
- SGBD tine datele intr-o structura numita **dictionarul de date al sistemului** sub forma de tabele pe care utilizatorii care au drepturile necesare le pot accesa printr-o serie de vederi sau sinonime.
- Fiecare vedere are un prefix al numelui ce defineste categoria respectiva
  1. Prefixul **USER\_** este comun utilizatorilor prin care se pot accesa informatii despre obiectele detinute de utilizator
  2. Prefixul **ALL\_** este specific vederilor ce contin informatii despre toate tabelele la care utilizatorul are acces
  3. Prefixul **DBA\_** este folosit pentru vederile accesibile doar utilizatorilor care su privilegii de administrator al bazei de date

## Cereri pe o singura tabela

- O cerere SQL este formata din mai multe clauze, fiecare clauza incepe cu un cuvant cheie. Prima clauza determina si tipul cererii.

`SELECT [DISTINCT] lisata de expresii`

`FROM nume_tabela`

`WHERE conditie_coloane`

`ORDER BY criterii_sortare_rezultat`

Efect:

- Se parcurg toate liniile tabelei invocate prin clauza FROM.
- Daca conditia din clauza WHERE este indeplinita se adauga liniile la rezultat. Cand WHERE lipseste sunt adaugate toate liniile.
- Coloanele rezultatului sunt date de lista expresiilor clauzei SELECT
- Fara DISTINCT nu se vor elimina duplicatele din rezultat
- Rezultatul este sortat functie de criteriile specificate in clauza ORDER BY. Daca lipseste ordinea este dependenta de ordinea din tabela initiala.

## ■ Invocarea constantelor

In select pot fi invocate constante: numerice, sir de caractere sau NULL.

Ex: **SELECT ‘Salariul persoanei ’, Nume, Prenume, ‘ este ’, Salariu  
FROM Angajat**

va genera o tabela cu 5 coloane in care in prima coloana se culege constanta Salariul persoanei si in coloana 4 constanta este, coloanele 2, 3 si 5 aduc informatii din tabela Angajat.

## ■ Expresii aritmetice

Elementele din lista SELECT pot fi expresii complexe continand functii si operatori. Operatori disponibili: \*, /, +, -.

Ex: **SELECT Nume, Prenume, Salariu, (Salariu + 100)\*1.1  
FROM Angajat**

Genereaza o tabela cu 4 coloane, coloana 4 fiind o coloana calculata dupa expresia precizata.

## ■ Expresii concatenate

Operatorul de concatenare (||) permite crearea de coloane ce culeg informatii dintr-o expresie ce contine constante si rezultate ale operatiilor efectuate asupra coloanelor.

Ex: `SELECT 'Salariul persoanei ' || Nume || ' ' || Prenume || ' este ' || Salariu || ' lei'`  
`FROM Angajat`

Genereaza o tabela cu un singur camp. In acest exemplu numele coloanei este dat de expresia din clauza SELECT. Aceasta expresie poate excede numarul de caractere permise pentru definirea unui camp si este necesara redefinirea sa prin introducerea de **ALIAS** la coloana.

Ex: `SELECT 'Salariul persoanei ' || Nume || ' ' || Prenume || ' este ' || Salariu || ' lei' AS Descriere`  
`FROM Angajat`

AS poate fi omis fara nici un efect asupra rezultatului:  
..... ‘lei’ Descriere

- Eliminarea duplicatelor

Pot fi obtinute inregistrari identice daca nu se invoca in clauza SELECT un camp cheie.

Ex: **SELECT Slalariu FROM Angajat**

Va genera inregistrari identice daca in companie sunt angajati cu acelasi salar. Utilizarea cuvantului cheie DISTINCT elimina duplicatele

Ex: **SELECT DISTINCT Slalariu FROM Angajat**

In toate cererile anterioare operatiile se executa pentru toate inregistrarile din tabela. Specificarea unei conditii de selectie poate fi realizata prin introducerea clauzei **WHERE expresie\_logica**, expresie ce este evaluata pentru fiecare inregistrare a tablei aducand in rezultat acele inregistrari pentru care rezultatul evaluarii este **TRUE**

Ex: **SELECT Nume, Prenume FROM Angajat WHERE D\_nr=4**

## ■ Operatori de comparatie la clauza WHERE

Operatori uzuali:

- Egal =
- Mai mare >
- Mai mare sau egal >=
- Mai mic <
- Mai mic sau egal <=
- Diferit <>      !=      ^=

## ■ Conditii compuse

Pot fi formulate conditii complexe cu operatorii logici AND, OR, NOT si paranteze de grupare pentru ordine operatii

Ex: (D\_nr = 3 and Salariu >400) or (D\_nr = 5 and Salariu\*1.1<=300)

- Operatorul BETWEEN

Operatorul specifica apartenența unei valori la un interval închis  
expresie BETWEEN valoare\_minima AND valoare\_maxima

Ex: SELECT Nume, Prenume, Salariu  
FROM Angajat

WHERE Salariu >300 and D\_nr BETWEEN 2 AND 5

- Operatorul IN

Operatorul testează apartenența unei valori la o mulțime statică  
expresie IN (val\_1, val\_2,...,val\_N)

EX: SELECT Nume, Prenume, Salariu  
FROM Angajat  
WHERE Salariu >300 and D\_nr IN (2, 4, 6)

In lista de valori a operatorului IN poate fi inclus și NULL, NOT NULL  
Operatorul poate fi negat NOT IN (lista\_valori)

- Operatorul LIKE

Se utilizeaza pentru a testa daca valoarea unei expresii respecta un anumit sablon

expresie LIKE 'SABLON' [ESCAPE 'caracter']

Sablonul se pune intre apostrofi si poate contine caracterele \_ si %:

- \_ inlocuieste un caracter
- % inlocuieste orice sir de caractere, inclusiv sir vid

Ex: **SELECT Nume, Prenume FROM Angajat**

WHERE UPPER(Prenume) LIKE 'A%A'

- Operatorul IS NULL

Valorile NULL au proprietati deferite, ele nu pot fi utilizate in expresii cu operatori de egalitate sau diferit. Pentru acestea se utilizeaza **IS NULL** si **IS NOT NULL**

Ex: **SELECT Nume, Prenume FROM Angajat**

WHERE sssn IS NULL

## ■ Clauza ORDER BY

ORDER BY specifica ordinea inregistrarilor in tabela produsa de cerere. Pot fi invocate atat nume de coloane ce apar in rezultat cat si nume de coloane ce nu apar in rezultat. Ordonarea se efectueaza dupa ordinea specificarii la clauza ORDER BY, ordinea default este ascendent (ASC), DESC specifica ordine descrescatoare.

Ex: `SELECT Nume, Prenume, D_nr, Salariu  
From Angajat  
WHERE D_nr IN (1,3,5) and Salariu >200 and Sssn IS NULL  
ORDER BY D_nr ASC, Salariu DESC, Nume`

Poate fi invocata drept criteriu de ordonare in ORDER BY si numarul coloanelor

Ex: `SELECT Nume, Prenume, D_nr, Salariu  
From Angajat  
WHERE D_nr IN (1,3,5) and Salariu >200 and Sssn IS NULL  
ORDER BY 3 ASC, 4 DESC, 1`

## 4. Cereri SQL pe mai multe tabele

- Aceste cereri implementeaza operatiile algebrei relationale produs cartezian si JOIN.
- Clauza FROM va contine o lista de tabele.
- Clauza WHERE va contine si conditii de corelare a inregistrarilor din diverse tabele numita si **conditie JOIN**.
- O conditie JOIN este numita **equjoin** daca se specifica o egalitate intre valorile coloanelor ce definesc conditia.
- Daca conditia nu specifica egalitate atunci se spune ca este **non-equjoin**.
- Daca coloanele ce participa la o operatie equjoin au acelasi nume se va numi **natural join**.
- Pot fi luate in consideratie si situatiile in care se adauga in rezultat si inregistrari ce nu indeplinesc conditia JOIN numita si **join extern (outer join)**.

- Produs cartezian

Operatia se produce atunci cand in clauza WHERE nu se specifica nici o conditie intre campurile tablelor.

Ex: `SELECT *`

```
FROM Angajat, Departament  
WHERE D_nr = 4
```

- Equjoin

Consideram doar combinarea inregistrarilor tablei Angajat cu cele din departament dar numai pentru departamentul din care face parte un anumit angajat.

```
SELECT Nume, Prenume, D_nume, D_location  
FROM Angajat, Departament  
WHERE D_nr = Dep_nr
```

Daca cele doua campuri au acelasi nume in tablele Angajat si Departament obtinem o operatie natural JOIN. Din punctul de vedere al rezultatului nu este nici o diferenta.

Sunt situatii atunci cand se fac operatii cu mai multe tabele ca o conditie sa devina ambigua datorita existentei campurilor cu acelasi nume. Tehnica uzuala este cea de utilizare ALIAS pentru tabela.

- Alias implicit – numele tabelei.

Ex: `SELECT Angajat.Nume, Angajat.Prenume, Departament.D_nume,  
Departament.D_location  
FROM Angajat, Departament  
WHERE Angajat.D_nr = Departament.Dep_nr`

- Alias explicit – nume asociat tabelei.

Ex: `SELECT A.Nume, A.Prenume, D.D_nume, D.D_location  
FROM Angajat A, Departament D  
WHERE A.D_nr = D.Dep_nr`

Un alias explicit este foarte util atunci cand se executa o operatie de JOIN intre o tabela si ea insasi (self-join). In acest caz aceeasi tabela este vazuta ca doua instante.

De exemplu obtinerea numelui si prenumelui fiecarui angajat si al supervisorului sau.

```
SELECT A.NUME,A.PREN,S.NUME, S.PREN  
FROM ANGAJAT A, ANGAJAT S  
WHERE A.SSN=S.SSN
```

Alte exemple

```
SELECT P_NR,DEP_NUME, NUME,INI,PREN,ADR,DAT_NA  
FROM PROIECT, DEPARTAMENT, ANGAJAT  
WHERE PROIECT.DEP_NR=DEPARTAMENT.DEP_NR AND  
DEP_MAN=SSN AND P_LOC='Brazi'
```

Cererea de listare pentru fiecare proiect localizat in 'Brazi' a numarului proiectului, numarului departamentului ce il coordoneaza, numelui, adresei si datei de nastere a managerului departamentului

```
SELECT      P_NR,DEP_NUME,NUME,INI,PREN,ADR,DAT_NA  
FROM        PROIECT P,DEPARTAMENT D,ANGAJAT  
WHERE       P.DEP_NR=D.DEP_NR AND DEP_MAN=SSN AND  
P_LOC='Brazi'
```

- Join extern (outher join)

Cererea de listare a numelor si prenumelor angajatilor impreuna cu supervisorii lor, iar pentru angajatii ce nu au supervisor numele si prenumele supervisorilor va fi NULL.

```
SELECT      A.NUME,A.PREN,S.NUME, S.PREN  
FROM        ANGAJAT A, ANGAJAT S  
WHERE       A.SSN (+)=S.SSN
```

Join extern poate fi invocat si cu alte conditii decat egalitate sau BETWEEN sau LIKE

## 5. JOIN SQL-3

De la versiunea 9i Oracle recunoaste si sintaxa SQL-3. Se remarcă faptul ca aceste operatii pot fi executate utilizand sintaxele descrise.

### ■ Clauza CROSS JOIN

Realizeaza produs cartezian. Spre deosebire de sintaxele anterioare a doua tabela nu se precizeaza in FROM ci in clauza CROSS JOIN

```
SELECT [DISTINCT] lista_expresii  
FROM tabela1  
CROSS JOIN tabela2
```

### ■ Clauza JOIN .. USING

Se utilizeaza cand in cele doua tabele invocate la join coloanele au acelasi nume, operatia fiind de tip equijoin

```
SELECT [DISTINCT] lista_expresii  
FROM tabela1  
JOIN tabela2 USING (nume_coloane)
```

- Clauza NATURAL JOIN

Este un caz particular al clauzei JOIN .. USING. Nu se mai specifica coloanele dupa care se face join si Oracle considera egalitatea intre valorile tuturor coloanelor cu acelasi nume.

```
SELECT [DISTINCT] lista_expresii  
FROM tabela1  
NATURAL JOIN tabela2
```

- Clauza JOIN .. ON

Clauza implementeaza un join general. Conditia si eventual conditiile suplimentare se pun la ON

```
SELECT [DISTINCT] lista_expresii  
FROM tabela1  
JOIN tabela2 ON (expresie_logica)
```

## ■ Clauza OUTER JOIN .. ON

Este un join extern cu sintaxa

```
SELECT [DISTINCT] lista_expresii  
FROM tabela1  
LEFT | RIGHT | FULL OUTER JOIN tabela2 ON (tabela1.nume_col  
= tabela2.nume_col)
```

Interpretare:

- LEFT – valorile nule provin din tabela 2
- RIGHT – valorile nule provin din tabela 1
- FULL – reuniunea dintre LEFT si RIGHT

**Obs:** Caracteristici rezultatul:

- Numar de coloane este egal cu numarul de expresii din clauza SELECT
- Numarul de linii este egal cu numarul liniilor ce indeplinesc conditia WHERE
- Parcurgerea liniilor este facuta pe baza serverului de baze de date si nu se garanteaza ordinea rezultatelor daca nu se invoca clauza ORDER BY.

O cerere simpla:

```
SELECT nume, prenume  
FROM Angajat
```

Returneaza numele si prenumele angajatilor companiei.

Daca se doreste afisare tuturor coloanelor se poate utiliza caracterul \*. Ordinea fizica este data de ordinea fizica a coloanelor tablei.

```
SELECT *  
FROM angajat
```

## Functii

Orice sistem de baze de date pune la dispozitie functii ce pot fi utilizate in cererile SQL

Functie de numarul de linii in care se calculeaza rezultatul se impart in doua mari categorii:

### 1. Functii pentru linia curenta

- Functii cu argumente numerice: argumente si rezultat numeric
- Functii pe siruri: argumente sir de caractere, rezultat sir sau numeric
- Functii cu argumente date calendaristice: argument data, rezultat data sau numar
- Functii de conversie tipuri data
- Functii cu argumente de diverse tipuri

### 2. Functii de grup, cunoscute si sub numele de functii statistice. Se pot aplica pe intreg continutul unei tabele sau numai pe anumite inregistrari

- MIN, MAX, AVG
- SUM

- COUNT
- STDDEV si VARIANCE pentru deviatia standard si varianta valorilor
  - A) [Functii pentru linia curenta](#)
  - Numarul de argumente este fixat prin definitia functiei
  - Un argument poate fi un nume de coloana, o constanta sau o pseudocoloana
  - Argumentul este o expresie ce contine un nume de coloana, constanta sau alta functie.
- 1. Functii cu argumente numerice
  - TRUNC(argument1 [,n]) intoarce un intreg daca n lipseste sau este 0, respectiv un numar cu n digits dupa punctul zecimal. Daca n este negativ truncherea se face la partea intreaga
  - ROUND(argument1[,n]) rotunjire la intreg sau la un numar de digits dupa punctul zecimal.
  - MOD(arg1, arg2) restul impartirii arg1 la arg2

Ex: `SELECT nume, prenume, TRUNC(salariu*1.1111, 2) Trunchiat,  
ROUND(salariu*1.1111, 2) Rotunjit, MOD(salariu*1.1111, 2) Rest  
FROM Angajat`

- `CEIL(arg)` si `FLOOR(arg)` intoarce cel mai mic intreg mai mare decat argumentul, respectiv cel mai mare intreg mai mic decat argumentul.

Ex: `CEIL(1.8) = 2;`      `FLOOR(1.8) = 1`

- Alte functii uzuale sunt cele aferente functiilor trigonometrice, logaritm, exponentiala, sign, power.

`ABS(arg_n), SIGN(arg_n), POWER(arg_n1,arg_n2), EXP(arg_n),  
SQRT(arg_n), LOG(arg_baza, arg_n), SIN(arg_radiani),  
COS(arg_radiani), TAN(arg_radiani), SINH(arg_n), COSH(arg_n),  
TANH(arg_n)`

Pentru toate aceste functii este necesar ca valorile argumentelor sa respecte domeniile de definitie ale functiilor.

## Functii cu argumente siruri de caractere

Argumentele sunt constante, nume de coloane, expresii ce returneaza sir de caractere. Daca un argument nu respecta aceasta conditie el este convertit automat, daca este posibil, la sir de caractere.

- LOWER(arg\_c), UPPER(arg\_c), INITCAP(arg\_c) conversie la litere mici, litere mari sau inceput litera mare.
- CONCAT(arg\_c1, arg\_c2) concatenare echivalent cu ||
- LENGTH(arg\_c)
- SUBSTR(arg\_c, start [,nr\_car]) sir incepand cu pozitia start cu lungimea nr\_car
- INSTR(arg\_c, subsir [,start]) pozitia subsirului in arg\_c de la inceput sau de la start
- LPAD(sir, dimensiune, [,subsir]), RPAD(sir, dimensiune, [,subsir]) umplere la stanga sau la dreapta pana la dimensiunea specificata cu subsirul specificat sau blank
- TRIM([LEADING|TRAILING|BOTH] [caracter] FROM arg\_c) eliminarea la stanga, dreapta sau ambele a caracterului specificat pana la intalnirea unui caracter diferit
- REPLACE(arg\_c, subsir [,subsir\_nou]) inlocuire subsir cu subsir\_nou

- `TRANSLATE(arg_c, sir1 [,sir2])` se inlocuieste caracterul corespondent din sir1 cu cel din sir2 in sirul dat de arg\_c
- `SOUNDEX(arg_c)` intoarce un cod alfanumeric al sonoritatii argumentului in limba engleza

Ex: Adaugarea la numele angajatilor din departamentul 5 a sufixului bucuresti daca lungimea numelui este mai mica de 5

```
SELECT CONCAT(nume, ' bucuresti') Nume_p  
FROM Angajat  
WHERE LENGTH(nume)<5
```

### Functii cu argumente date calendaristice

La stocarea datelor cele mai multe medii de baze de date pastreaza: secol, an, luna, zi, ora, minut, secunda. Pot fi efectuate operatii:

- Diferenta a doua date calendaristice
- Adaugarea unui numar la o data calendaristica
- Scaderea unui numar dintr-o data calendaristica
- Adunare si scadere cu o constructie “/24”

Principalele functii:

- ROUND(data [, ‘MONTH’ | ‘YEAR’])
- TRUNC(data [, ‘MONTH’ | ‘YEAR’])
- NEXT\_DAY(data [, numar zi | ‘nume\_zi’])
- ADD\_MONTH(data, numar\_luni)

### Functii de conversie

O serie de conversii sunt efectuate automat daca este posibil. Sunt posibile conversii fortate prin utilizare functiilor:

- TO\_CHAR(numar[ ,’sablon’])
- TO\_NUMBER (sir[ ,’sablon’])
- TO\_DATE(sir[ ,’sablon’])

## Functii cu argumente de tipuri diferite

Tipul parametrilor si numarul lor poate fi diferit de la apel la apel.

- NVL(expr1, expr2) intoarce expr1 daca este nul si expr2 altfel
- NVL2(expr1, expr2, expr3) intoarce expr2 daca expr1 este nul si expr3 altfel
- COALESCE(lista expresii) intoarce prima valoare nul din lista de expresii
- GREATEST(lista expresii) intoarce cea mai mare valoare din lista de argumente
- LEAST(lista expresii) intoarce cea mai mica valoare din lista de expresii
- DECODE(expr, test1, val1, test2, val2,...., testn, valn)
- CASE expresie WHEN test1 THEN rez1 [WHEN test2 THEN rez2..  
..... [ELSE rez\_implicit] ]  
END

## Functii statistice pe grupuri

Fiecare functie statistica calculeaza rezultatul pornind de la o multime de valori luand in consideratie sau nu valorile duplicate. Tratare:

- Cu exceptie COUNT(\*) toate celelalte ignora valorile nule;
- Implicit, se considera toate valorile dupicate adica ALL, pentru valorile distincte expresia va fi prefixata de cuvantul cheie DISTINCT

MIN([ALL|DISTINCT] expresie)

MAX([ALL|DISTINCT] expresie)

Ex: **SELECT Nume, Prenume, MIN(Salariu) minim, MAX(salariu) maxim FROM Angajat**

SUM ([ALL|DISTINCT] expresie)

AVG([ALL|DISTINCT] expresie)

Ex: **SELECT SUM(Salariu) suma1, SUM(DISTINCT Salariu) suma2,  
AVG(salariu) media1, AVG(DISTINCT Salariu) media2  
FROM Angajat**

**WHERE D\_nr=2 or D\_nr=4**

COUNT(\*) sau COUNT([ALL|DISTINCT] expresie) in care:

- COUNT(\*) intoarce numarul de linii
- COUNT(expresie) intoarce numarul de valori nenule ale expresiei
- COUNT(DISTINCT exprtesie) numarul de valori nenule si distincte

STDDEV([ALL|DISTINCT] expresie) deviatia standard

VARIANCE([ALL|DISTINCT] expresie) varianta valorilor

## Clauza GROUP BY

Permite calcularea valorilor statistice pe grupuri de inregistrari.

GROUP BY expr1 [,expr2, expr3, ....]

Ex: SELECT D\_nr, MAX(Salariu), MIN(Salariu), AVG(Salariu)

FROM Angajat

GROUP BY D\_nr

ORDER BY D\_nr

## Clauza HAVING

Permite filtrarea rezultatelor la nivel de grup. O conditie de filtrare la nivel de grup, adica o conditie pe rezultatul functiilor nu poate fi pusa in clauza WHERE ci numai in clauza HAVING

```
SELECT D_nr, MAX(Salariu), MIN(Salariu), AVG(Salariu)
      FROM Angajat
      GROUP BY D_nr
      HAVING AVG(Salariu) >420
      ORDER BY MIN(Salariu)
```

Obs: Functiile agregat sau de grup pot fi utilizate in cereri complexe cu Join.

```
SELECT Nume, Prenume, SUM(Ore)
      FROM Angajat, Lucreaza
      WHERE Angajat.ssn = Lucreaza.ssn
      GROUP BY Lucreaza.ssn
      HAVING SUM(Ore) < 40
```

## Subcereri

O subcerere este o cerere SELECT inclusa intr-o alta cerere SQL.

Aceste constructii se folosesc atunci cand rezultatul dorit nu se poate obtine cu o singura parcurgere a datelor. De exemplu daca se doreste obtinerea angajatului care lucreaza cel mai mare numar de ore la proiecte este necesara obtinerea maximului de ore dupa care se obtin datele despre angajat.

Subcererile pot sa apară în urmatoarele clauze:

- WHERE și HAVING ca expresii logice;
- ORDER BY în care ordonarea se face după rezultatul unei subcereri
- SELECT valoare prezenta în rezultatul final;
- FROM rezultatul este asimilat cu o tabelă temporară ce participă la cererea principală

Subcererile pot întoarce:

- O singura valoare
- O coloana
- O tabela

- Subcereri care intorc o singura valoare

Ex: `SELECT Nume, Prenume`

`FROM Angajat, Lucreaza`

`WHERE Angajat.ssn=Lucreaza.ssn and Ore = (SELECT Max(Ore)`  
`FROM Lucreaza)`

Obs: Subcererea trebuie sa fie in partea dreapta a expresiei logice evaluate

Pot fi utilizate mai multe subcereri in aceeasi cerere.

Se poate utiliza operatorul BETWEEN sau LIKE

Ex: `SELECT Nume, Prenume`

`FROM Angajat, Lucreaza`

`WHERE Angajat.ssn=Lucreaza.ssn and Ore BETWEEN (SELECT`  
`Avg(Ore)*0.7 FROM Lucreaza and SELECT Avg(Ore)*1.3 FROM`  
`Lucreaza)`

## ■ Subcereri care intorc o coloana

Coloanele introarse de subcerere sunt asimilate unei multimi. Conditia trebuie sa foloseasca operatorul IN sau negatul sau NOT IN.

Ex: `SELECT Nume, prenume  
 FROM Angajat, Lucreaza  
 WHERE Angajat.ssn=Lucreaza.ssn and  
 P_nr IN (SELECT P_nr FROM Proiect WHERE D_nr =3)`

Determina numele si prenumele angajatilor care lucreaza la proiecte coordonate de departamentul cu numarul 3.

`SELECT Nume, prenume  
 FROM Angajat, Lucreaza  
 WHERE Angajat.ssn=Lucreaza.ssn and  
 P_nr IN (SELECT P_nr  
 FROM Proiect, Departament  
 WHERE Departament.D_nr =Proiect.D_nr and  
 Departament.D_nume LIKE "Cercetare")`

Determina numele si prenumele angajatilor care lucreaza la proiecte coordonate de departamentele cu numele Cercetare.

Obs: Pot fi utilizati operatori de prefixare SOME, ANY, ALL pentru a interpreta operatia de comparatie.

- SOME si ANY conditie adevarata daca macar o valoare din cele returnate de subcerere verifica comparatia;
- ALL conditia este adevarata daca toate valorile returnate de subcerere verifica comparatia.

SELECT Nume, prenume

FROM Angajat, Lucreaza

WHERE Angajat.ssn=Lucreaza.ssn and

Ore > ALL (SELECT Ore

FROM Proiect, Lucreaza

WHERE Proiect.D\_nr=4 and Lucreaza.P\_nr =  
Proiect.P\_nr)

In acest caz se obtin angajatii care lucreaza un numar mai mare de ore decat toate orele prestate la proiectele coordonate de departamentul cu numarul 4

- Subcereri care intorc o tabela

Atunci cand o cerere intoarce un rezultat care are mai multe coloane el este asimilat cu o multime de linii si poate fi utilizat operatorul IN astfel:

**WHERE (lista\_expresii) IN (subcerere)**

Obs:

- Lista trebuie incadrata intre paranteze rotunde;
- Numarul de coloane din subcerere trebuie sa fie egal cu numarul de expresii;
- Corespondenta dintre valorile expresiilor si numarul de coloane este pozitionala;
- Trebuie respectat tipul elementelor corespunzator (eventual conversie automata)
- Rezultatul subcererii vid duce la conditie evaluata fals

Ex: Determinarea angajatilor ce lucreaza la un singur proiect

SELECT Nume, Prenume, Ore

FROM Angajat, Lucreaza

WHERE Angajat.ssn=Lucreaza.ssn and

(ssn, ore) IN (SELECT ssn, sum(ore)

FROM Lucreaza

GROUP BY Ssn)

In acest caz numarul de ore este egal cu suma orelor prestate de un angajat. O valoare NULL a unui camp duce la evaluare fals.

- Subcereri in HAVING

Expresiile logice cuprinzand subcereri pot fi utilizate si in clauza HAVING in acelasi mod. Singura restrictie consta in faptul ca aici conditiile vor contine doar elemente ce pot sa apara intr-o astfel de clauza: constante, expresii de grupare, functii statistice.

Ex: Afisarea numelui, prenumelui si numarului minin si maxim de ore desfasurate de angajati la proiecte, pentru proiectele la care media numarului de ore este peste media calculata la toate proiectele companiei.

```
SELECT Nume, Prenume, MIN(Ore), MAX(Ore)
      FROM Angajat, Lucreaza
     WHERE Angajat.ssn = Lucreaza.ssn
       GROUP BY P_nr
      HAVING AVG(Ore) > (SELECT AVG(ore) FROM Lucreaza)
```

Ex2: Afisare numelui si mediei orelor pentru proiectele cu cea mai mare medie de ore

```
SELECT P_nume, AVG(Ore)
      FROM Lucreaza
       GROUP BY P_nr
      HAVING AVG(Ore) >= (SELECT MAX(AVG(ore)) FROM Lucreaza)
       GROUP BY P_nr)
```

- Subcereri in clauza FROM

Daca o subcerere apare la clauza FROM ea va fi tratata ca o tabela temporara

Ex: Determinarea angajatilor care lucreaza la proiectele coordonate de departamentul cu numarul 5 intre 6 si 20 ore.

**SELECT Nume, Prenume**

```
FROM (SELECT * FROM Angajat A, Lucreaza B, Proiect C  
      WHERE A.ssn = B.ssn and B.P_nr = C.P_nr and C.Dep_nr = 5  
      and Ore BETWEEN 6 and 20)
```

Obs:

- Daca mai multe tabele sunt invocate in cererea principala unei subcereri i se poate asocia un alias pentru a elimina ambiguitatea.
- Subcererile sunt tratate similar cu tabelele permanente putand fi implicate si in operatii SQL-3

- Subcereri corelate

La exemplele anterioare cererea se executa o singura data dupa care rezultatul este utilizat la evaluare. Sunt situatii in care rezultatul unei subcereri este dependent de valorile liniei curente, numite si subcereri corelate.

Ex: Sa se afiseze angajatii care lucreaza la un proiect un numar de ore peste media numarului de ore lucrat de angajatii la acelasi proiect.

SELECT Nume, Prenume, Ore

FROM Angajat A, Lucreaza B

WHERE A.ssn = B.ssn and Ore > (SELECT AVG(ore)

FROM Lucreaza

WHERE B.P\_nr=P\_nr)

Pentru fiecare inregistrare valoarea P\_nr este transmisa subcererii prin B.P\_nr calculand punctajul mediu daca B.P\_nr=P\_nr. Daca Ore este mai mare decat media se va include in rezultat.

- Operatorul EXISTS

Operatorul testeaza daca subcererea primita ca argument intoarce rezultat nevid.

Ex: Sa se obtina angajatii companiei ce au persoane in intretinere

**SELECT Nume, Prenume**

**FROM Angajat**

**WHERE EXISTS (SELECT \***

**FROM Intretinut**

**WHERE Angajat.ssn = ssn)**

- Subcereri pe clauza ORDER BY

In versiunile noi clauza ORDER BY poate contine o subcerere corelata ce intoarce o singura valoare.

O cerere necorelata nu duce la ordonarea rezultatului deoarece rezultatul fiind o constanta are aceeasi valoare pentru fiecare linie supusa sortarii.

Ex: Sa se obtina angajatii companiei ordonati descreasator dupa numarul persoanelor aflate in intretinere

SELECT Nume, Prenume

FROM Angajat

ORDER BY (SELECT COUNT(\*) FROM Intretinut

WHERE Angajat.ssn=Intretinut.ssn) DESC

- Subcereri pe clauza SELECT

Ca si la cele din clauza ORDER BY aceste subcereri trebuie sa intoarca o singura valoare

Ex: Sa se obtina numele angajatilor si numarul persoanelor aflate in intretinere ordonati crescator dupa numarul persoanelor in intretinere.

SELECT Nume, (SELECT COUNT(\*) FROM Intretinut

WHERE Angajat.ssn=Intretinut.ssn)

FROM Angajat

ORDER BY (SELECT COUNT(\*) FROM Intretinut

WHERE Angajat.ssn=Intretinut.ssn)

## ■ Operatorii UNION, INTERSECT, MINUS

Acesti operatori permit combinarea prin REUNIUNE, INTERSECTIE, DIFERENTA a rezultatelor mai multor cereri. In acest caz coloanele sunt denumite de argumentele primei cereri SELECT, trebuie sa aiba acelasi numar de coloane si tipuri de date pozitionale compatibile. ORDER BY poate fi utilizata doar la sfarsitul frazei, nu si la cererile componente

Ex: Sa se obtina angajatii care lucreaza in departamentul cu numarul 3, precum si cei din departamentul cu numarul 5 toti avand salariu mai mare de 400

SELECT Nume, Prenume FROM Angajat WHERE D\_nr=3

UNION

SELECT Nume, Prenume FROM Angajat WHERE D\_nr=5

INTERSECT

SELECT Nume, Prenume FROM Angajat WHERE Salariu>400

ORDER BY Nume

Obs: Operatorii sunt in general utilizati atunci cand rezultatele provin din mai multe tabele.



# Alte obiecte in BD

## Preambul

In aceasta sectiune vom trata:

- Adaugarea liniilor intr-o tabela
- Stergerea liniilor dintr-o tabela
- Modificarea valorilor stocate intr-o tabela
- Mecanismul tranzactiilor si consistenta la citire
- Instructiunile specifice de realizare sunt:
  1. INSERT
  2. UPDATE
  3. DELETE
  4. MERGE
  5. COMMIT, ROLLBACK
  6. SAVEPOINT
- Vederi, indexuri, sechete.

## Inserarea liniilor

Sunt posibile doua tipuri de comenzi SQL pentru inserarea de date într-o tabelă:

- INSERT
- MERGE (combinatie INSERT/UPDATE)

### 1. Inserarea liniilor prin specificarea valorilor

`INSERT INTO nume_tabela [(lista_coloane)]  
VALUES (lista_valori)`

Dacă nu se precizează lista coloanelor avem urmatorul efect:

- O linie “completa” este inserată
- Numărul valorilor din VALUES este egal cu numărul de coloane
- Ordinea valorilor și tipul de date este același cu cel al coloanelor în corespondență pozitională
- Dacă valoarea unei coloane este nula se specifică NULL
- Dacă o coloană are valoare implicită se specifică DEFAULT, altfel valoarea
- Fiecare linie de inserat necesită o cerere separată

Ex: `INSERT INTO Lucreaza VALUES ('1561124400273', 27, 13)`

`INSERT INTO Lucreaza VALUES ('1561124400273', 27, NULL)`

`INSERT INTO Lucreaza VALUES ('1561124400273', DEFAULT, 15)`

Daca se specifica o lista de coloane se poate insera o linie

“incompleta”, coloanele neprecizate vor avea valoarea DEFAULT daca aceasta exista sau valoare NULL in caz contrar. Numarul coloanelor din lista trebuie sa fie egal cu numarul de valori in corespondenta pozitionala.

Ex: `INSERT INTO Angajat (Nume, Prenume, ssn, D_nr) VALUES ('Popescu', 'Ion', '1541124400273', 4)`

Obs:

- Daca o coloana nu respecta constrangerile din definirea tablei operatia va esua.
- Rezultatul unei cereri necorelate care intoarce o singura valoare poate fi folosita in lista de valori

Ex: `INSERT INTO Departament (D-nr) VALUES((SELECT MAX(D_nr)  
FROM Departament)+1)`

## 2. Inserarea rezultatului unei cereri

**INSERT INTO nume\_tabela [(lista\_coloane)] cerere\_select**

Descriere:

- Se executa cererea SELECT
- Liniile rezultatului sunt inserate in tabela
- Numarul coloanelor si numarul valorilor este egal si au tip de data identic pozitional
- Daca o linie a rezultatului nu respecta constrangerea de integritate intreaga comanda va esua, nu se insereaza nimic.

Ex: **INSERT INTO Dep\_info (Dep\_nr, Nr\_ang, Medie\_sal)**

**SELECT D-nr, Count(ssn), AVE(Salariu)**

**FROM Angajat**

**GROUP BY D\_nr**

Obs: In locul unui nume de tabela poate fi folosita o subcerere, asa cum se va discuta la paragraful WITH CHECK OPTION

## Stergerea liniilor

Sintaxa:

```
DELETE FROM nume_tabela  
[WHERE conditie]
```

In clauza WHERE pot aparea acelasi conditii ca la operatiile SELECT si chiar subcereri

Daca WHERE nu exista se sterg toate liniile tabelei.

Ex: `DELETE FROM Angajat WHERE D_nr=5`

```
DELETE FROM Angajat  
WHERE ssn IN (SELECT ssn FROM Lucreaza  
WHERE Ore BETWEEN 2 and 10)
```

Obs: Cererea va returna o eroare daca una dintre liniile ce indeplinesc conditia contine o cheie referita printr-o constrangere de tip FOREIGN KEY intr-o tabela din baza de date si optiunea de manipulare a constrangerii nu permite stergerea

# Actualizarea datelor

Sintaxa:

```
UPDATE nume_tabela  
    SET col1=expresie1 [, col2=expresie2, .... ]  
    [WHERE conditie]
```

Obs:

- In WHERE pot sa para aceleazi conditii ca la operatiile SELECT
- Daca WHERE lipseste se actualizeaza toate liniile, altfel numai cele care indeplinesc conditia specificata
- O coloana nu poate sa apara in SET de mai multe ori

Ex: UPDATE Angajat SET Salariu=Salariu\*1.1 WHERE D\_nr=4

UPDATE Angajat SET Adresa = DEFAULT

UPDATE Angajat SET Functia = NULL

In clauza SET si WHERE se pot folosi subcereri

Ex: UPDATE Dep\_info

```
SET Nr_ang = (SELECT Count(ssn) FROM Angajat WHERE  
D_nr=Dep_info.Dep_nr)  
WHERE Dep_nr IN (SELECT D_nr FROM Angajat GROUP BY  
D_NR HAVING count(*) >3)
```

Obs: 1. Atentie clauza WHERE

2. Anumite actualizari esueaza prin nerespectarea conditiilor de integritate:

- Noua valoare nula si exista o constrangere NOT NULL
- Noua valoare pentru un camp cheie primara sau valoare unica se regaseste in tabela
- Noua valoare nu verifica o conditie CHECK
- Vechea valoare este referita prin constrangeri de tip FOREIGN KEY
- Noua valoare nu se regaseste in tabela referita prin FOREIGN KEY

## Cereri MERGE

Cererea efectueaza fie actualizarea fie inserarea unor linii intr-o tabela dupa cum indeplineste sau nu conditia de join. Sintaxa:

```
MERGE INTO nume_tabela1 [alias tabela1]
  USING nume_tabela2 | nume_vedere | subcerere
  ON (conditie join)
  WHEN MATCHED THEN
    UPDATE SET col1=expr1 [,col2=expr2, ....]
  WHEN NOT MATCHED THEN
    INSERT [(lista_coloane)] VALUES (lista_valori)
```

Ex: MERGE INTO Dep\_info

```
  USING (SELECT D_nr, COUNT(*) AS Nr FROM Angajat GROUP
         BY D_nr)
  ON (Dep_nr=D_nr)
  WHEN MATCHED THEN
    UPDATE SET Dep_info.nr_ang = Nr
  WHEN NOT MATCHED THEN
    INSERT VALUES (D_nr, Nr)
```

## Consistenta la citire, blocari implicite, tranzactii

Sistemele de baze de date gestioneaza accesul mai multor utilizatori la aceleasi tabele. Operatiile cerute de acestia se impart in:

- Operatii de citire – realizate prin SELECT
- Operatii de scriere – INSERT, UPDATE, DELETE, MERGE

Atunci cand utilizatorii fac numai operatii de citire nu sunt probleme de consistenta, ei au acces la aceeasi informatie simultan.

Cand unul dintre utilizatori face operatii de scriere in tabele sunt necesare mecanisme de asigurarea consistentei.

- Mecanismul uzual este cel de blocare implicita la scriere si serializare, modificarile nu sunt vizibile pana la scrierea definitiva in baza de date prin COMMIT.
- Modificarile efectuate inainte de COMMIT pot fi revocate
- La revocare si la COMMIT sunt ridicate toate blocarile.

Sa analizam urmatoarea succesiune de cereri la aceeasi tabela in ordine temporara:

- T1 U1 UPDATE la o linie
- T2 U1 Vizualizare (vede modificarile facute prin UPDATE)
- T3 U2 Vizualizare (nu vede modificarile cerute de U1)
- T4 U3 UPDATE la aceeasi linie
- T5 U1 Executa COMMIT (datele sunt scrise permanent) si blocare de cererea lansata de U3
- T6 U1 Vizualizare (nu vede ce vrea sa faca U3 deoarece acesta nu a executat COMMIT)

## Tranzactii

O tranzactie este o succesiune de operatii SQL-DML care poate fi incheiata prin scrierea in baza sau revocata. Operatiile DDL si DCL nu sunt vazute ca tranzactii si nu pot fi revocate.

Reguli:

- La inchiderea oricarei sesiuni de lucru se face COMMIT automat
- Înainte de COMMIT o tranzacție poate fi revocată total sau parțial
- La caderea tensiunii de alimentare sistemul face automat ROLLBACK
- O cerere poate determina COMMIT automat dacă sistemul este setat corespunzător (SET AUTOCOMMIT ON). Default este OFF.

Comenzi aditionale

- Marcarea punctelor de revenire prin **SAVEPOINT nume**
- Revocare parțială sau totală prin  
**ROLLBACK TO [SAVEPOINT] nume**

Într-o tranzacție pot fi puse oricate puncte de revenire. Acestea trebuie să aibă nume distinct.

## Vederi

- Vederile materializate sunt tabele create prin interogare si sunt construite pe baza datelor din baza.
- Vederile nu sunt salvate ca tabele ci doar ca definitii SQL.
- Invocarea unui view determina executarea cererii SQL ce produce continutul prin recalculare, datele sunt actualizate.
- In view nu pot fi definite constrangeri de integritate, pot exista constrangeri mostenite de la tabelele pe baza fcarora vederea este construita
- In view pot fi definite restrictii de acces la date ( read only)

Sintaxa:

```
CREATE [OR REPLACE | FORCE | NOFORCE] VIEW Nume_vedere  
[(lista coloane)] AS subcerere  
[WITH CHECK OPTION [CONSTRAINT Nume_constrangere]]  
[WITH READ ONLY [CONSTRAINT Nume_constrangere]]
```

In care:

- Nume\_vedere specifica numele asociat vederii create (**Atentie la nume existent**)
- Lista\_coloane specifica numele coloanelor din vedere. Daca lista lipseste, coloanele vederii mostenesc numele coloanelor din cererea SQL
- Daca rezultatul cererii asociate vederii au nume ce nu respecta restrictiile de definire a numelor coloanelor acestea trebuie fie redefinite, fie se adauga alias
- OR REPLACE – daca vederea cu acel nume exista ea este inlocuita cu noua definitie. Daca vederea exista si nu se utilizeaza OR REPLACE se semnaleaza o eroare
- FORCE|NOFORCE (implicit NOFORCE) – la crearea unei vederi sistemul verifica existenta tabelelor invocate si semnaleaza erori daca vreuna dintre tabele nu exista sau cererea nu se poate executa. Cu optiunea FORCE nu se face verificarea si se stocheaza definitia vederii (**Atentie la utilizare vedere**).
- WITH CHECK OPTION impiedica efectuarea de modificari in baza daca liniile asociate/actualizate nu sunt regasite prin cererea asociata vederii. Aceasta optiune este similara cu o constrangere si ii poate fi asociat un nume

- WITH READ ONLY optiunea specifica faptul ca nu sunt permise modificari ale datelor prin intermediul vederii.

Ex:

```
CREATE OR REPLACE VIEW Dep_info AS  
SELECT D-nr AS Dep_nr, COUNT(*) AS Nr_ang, AVG(Salariu)  
Medie_sal FROM Angajat WHERE Salariu >200 GROUP BY D_nr  
WITH CHECK OPTION WITH READ ONLY
```

Obs:

- O vedere poate fi utilizata pentru crearea altrei vederi.
- Chiar daca nu exista optiunea READ ONLY optiunea CHECK nu va permite modificarea datelor din baza daca liniile nu sunt regasite in vedere
- Optiunea READ ONLY nu permite modificarea datelor prin vedere
- O vedere poate fi stearsa prin cererea **DROP VIEW Nume\_vedere**
- Stergerea unei vederi nu poate fi revocata prin ROLLBACK deoarece este o cerere DDL

## Modificarea datelor prin vederi

- O vedere este doar o definitie si ca urmare modificarea datelor prin view se realizeaza in tabelele din baza specificate in definitia vederii
- Mecanismele de tranzactie in view sunt identice cu cele descrise pentru tabele
- Cererea DELETE poate fi aplicata unei vederi numai daca:
  1. Are la baza o singura tabela
  2. Nu contine functii de grup
  3. Nu contine clauza GROUP BY
  4. Nu contine particula DISTINCT
  5. Nu contine functia ROWNUM, functie care intoarcе numarul de ordine al liniilor in rezultat, in ordinea in care acestea au fost obtinute de sistem, inainte ca ele sa fie ordonate de o eventuala clauza ORDER BY

Ex: vederea

```
CREATE OR REPLACE VIEW Sterge AS  
SELECT Nume, Prenume, D_nr, Salariu FROM Angajat WHERE  
D_nr IN (2,4,5)
```

Stergere: Stergere valida, sterge inregistrarile din tabela angajat daca salariul este mai mic decat 200 si D\_nr este 2,4 sau 5:

```
DELETE FROM Sterge  
WHERE Salariu <200
```

UPDATE poate fi realizat prin intermediul unei vederii daca sunt respectate toate restrictiile anterioare si nu se modifica coloane definite prin expresii.

Ex. La modificarea prin vederea Modif nu poate fi modificata avand conditie pe coloana Sal\_marit intrucat a fost obtinuta printr-o expresie

```
CREATE OR REPLACE VIEW Modif AS  
SELECT Nume, Prenume, D_nr, Salariu*1.5 Sal_marit FROM  
Angajat WHERE D_nr IN (2,4,5)
```

**INSERT prin view** poate fi realizat daca:

- Vederea verifica toate restrictiile definite anterior pentru stergere
- Vederea contine toate coloanele tableei de baza pentru care exista constrangeri de tip NOT NULL
- Comanda de inserare nu incearca sa insereze date in coloane care provin din expresii

Obs finale:

- Prin vederi se permit accesul numai la anumite date pentru utilizatori, vederea fiind calculata pentru fiecare utilizator
- Orice modificare prin vederi, daca este valida, se executa in tabelele ce definesc vederea cu respectarea constrangerilor impuse acestor tabele
- Modificarile prin vederi se pot efectua numai daca se respecta restrictiile aferente tipului de modificare.

## Secvente

In multe aplicatii este nevoie de generarea unor secvente de numere care sa fie utilizate in comenzi de tip INSERT sau UPDATE.

O solutie de calcul al valorii maxime si a incrementarii acesteia nu este corecta in contextul accesului concurrent. De ex:

```
SELECT MAX(D_nr) +1  
FROM Angajat
```

Nu garanteaza obtinerea unei valori unice. Solutia: definirea de secventa asociata tablei. Sintaxa:

```
CREATE SEQUENCE nume_secventa  
[INCREMENT BY pas]  
[START WITH val_initiala]  
[MAXVALUE val_maxima]  
[MINVALUE val_minima]  
[CYCLE |NOCYCLE]  
[CACHE numar_valori | NOCACHE]
```

Obs:

- Majoritatea optiunilor pot lipsi
- Pasul poate fi si negativ
- CYCLE produce ciclarea la atingerea valorii maxime sau minime
- CACHE specifica numarul urmatoarelor valori calculate aprioric

Ex:

```
CREATE SEQUENCE dep_seventa
INCREMENT BY 2
START WITH 20
MAXVALUE 100
CACHE 18
```

Fiecare seventa se apeleaza prin intermediul a doua pseudocoloane:

- Nume\_seventa.NEXTVAL – urmatoarea valoare
- Nume\_seventa.CURRVAL – ultima valoare utilizata

```
INSERT INTO Departament VALUES ('Certcetare',  
    dep_secventa.NEXTVAL, ....)
```

O secventa poate fi modificata prin:

```
ALTER SEQUENCE nume_secventa  
[INCREMENT BY pas]  
[START WITH val_initiala]  
[MAXVALUE val_maxima]  
[MINVALUE val_minima]  
[CYCLE |NOCYCLE]  
[CACHE numar_valori | NOCACHE]
```

Stergerea unei secente:

```
DROP SEQUENCE nume_secventa
```

Stergerea unei secente nu poate fi revocata prin ROLLBACK

Un index este o structura de cautare rapida utilizat pentru cresterea vitezei de executare a cererilor. Oracle creaza automat indecsi pentru cheile specificate prin constrangerile PRIMARY KEY si UNIQUE.

Sintaxa:

```
CREATE INDEX nume_index  
ON nume_tabela (expresie1 [, expresie2, .....])
```

Ex:

```
CREATE INDEX ang_index  
ON Angajat (D_nr, Data_n DESC)  
Stergere: DROP INDEX nume_index
```

Nu totdeauna indexarea duce la cresterea vitezei. Sunt situatii in care se recomanda utilizarea indexilor, altele in care aceasta nu este recomandata.

Indecsi sunt cu automentinere, adica dupa modificari ale datelor el reflecta acele modificarile.

Cazuri in care se recomanda indexarea:

- Coloanele de indexare contin un procent mare de valori nule
- Coloanele de indexare sunt utilizate in clauza WHERE a unor cereri executate frecvent
- Coloanele de indexare contin o mare varietate de valori
- Tabelele contin un mare numar de linii si majoritatea cererilor afecteaza maxim 4% din inregistrari.

Nu se recomanda indexarea:

- Tabela are un numar mic de linii
- Majoritatea cererilor afecteaza un numar mai mare de 4% din inregistrari
- Coloanele de indexare nu sunt utilizate in clauza WHERE a unor cereri executate frecvent
- Tabela este frecvent modificata prin INSERT, UPDATE, DELETE
- Conditiiile WHERE contin expresii calculate pe baza coloanelor de indexare

## Sinonime

Obiectele bazei de date pot avea asociate nume alternative numite **sinonime** avand ca scop:

- Nume alternative mai sugestive;
- Nume alternative mai scurte;

Sintaxa:

**CREATE [PUBLIC] SYNONIM nume\_sinonim**

**FOR nume\_object**

Ex:

**CREATE PUBLIC SYNONIM Ang FOR Angajat**

Stergere sinonim:

**DROP [PUBLIC] SYNONIM nume\_sinonim**

**Obs: Foarte important pentru baze de date distribuite pentru transparenta localizarii datelor**

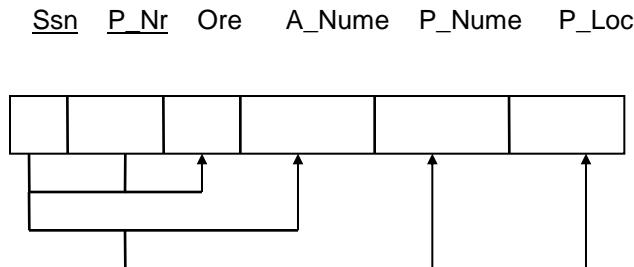


# Normalizare si securitate BD

## Dependente functionale

- Un concept important in constructia schemelor relatie este **dependenta functională**. Dependenta functională se exprima ca o restrictie intre doua seturi de atribută intr-o baza de date.
- **Dependenta functională**, notata prin  $X \rightarrow Y$  intre doua seturi de atribută  $X$  si  $Y$ , ce sunt subseturi ale lui  $R$  specifică o restrictie a n-uplurilor posibile ce formează o relatie instantă  $r$  în  $R$ . Restrictia statuează ca pentru orice două n-upluri  $t_1$  și  $t_2$  în  $r$  pentru care  $t_1(X) = t_2(X)$  va trebui să implice că  $t_1(Y) = t_2(Y)$ .
- Se poate spune că există o dependență funcțională de la  $X$  la  $Y$  sau că  $Y$  este *functional dependent* de  $X$ . Funcțională dependență este notată prin FD; setul atributelor  $X$  este numit și partea stanga a lui FD iar setul atributelor  $Y$  partea dreapta a FD.
- dacă o restrictie în  $R$  statuează că nu poate fi mai mult de un n-uplu cu o valoare data  $X$  în orice relatie instantă  $r(R)$ , adică  $X$  este o cheie candidată în  $R$ , atunci  $X \rightarrow Y$  pentru orice subset al atributelor  $Y$  din  $R$ .
- dacă  $X \rightarrow Y$  în  $R$ , nu se poate spune numeric despre implicatia  $Y \rightarrow X$  în  $R$ .

- Se considera schema relatie *ANGAJAT\_PROJECT* pentru care de la semantica atributelor, se cunoaste ca urmatoarea dependenta functionala a atributelor relatiei trebuie respectata.
- $\text{Ssn} \rightarrowtail \text{A\_Nume}$
- $\text{P\_Nr} \rightarrowtail \{\text{P\_Nume}, \text{P\_Loc}\}$
- $\{\text{Ssn}, \text{P\_Nr}\} \rightarrowtail \text{Ore}$



Reguli de inferenta:

Se noteazat prin  $F$  un set de dependente funktionale ce sunt specificate in schema relatiei  $R$ . Tipic, proiectantul bazei de date va specifica dependentele funktionale ce sunt evidente din punct de vedere semantic. Sunt insa o serie de alte dependente funktionale ce vor fi tinute in toate relatiile instanta ce satisfac dependentele in  $F$ . Setul tuturor dependentele funktionale este numit si inchiderea lui  $F$  si se noteaza cu  $F^+$ .

Ca exemplu, se presupune ca se specifica urmatorul set al dependentelor functionale F pentru schema relatie:

$$F = \{Ssn \rightarrow \{A\_Nume, Dat\_Na, Adr, D\_Nr\}, \\ D\_Nr \rightarrow \{D\_Nume, Dep\_Manager\}\}$$

Se pot inferentia urmatoarele dependente functionale aditionale din F:

$$Ssn \rightarrow \{D\_Nume, Dep\_Manager\}, \\ Ssn \rightarrow Ssn, \\ D\_Nr \rightarrow D\_Nume$$

Atunci cind se discuta de dependenta functionala se va utiliza un set de notatii in care concatenarea atributelor se va reprezenta simplu eliminind virgula, asa ca:

FD: [X,Y]  $\rightarrow$  Z va fi scrisa XY  $\rightarrow$  Z, si

FD: [X,Y,Z]  $\rightarrow$  [U,V] se va scrie XYZ  $\rightarrow$  UV.

Regulile de dependenta functionala pot fi sintetizate in urmatoarea forma:

- (*Regula de reflexivitate*) Daca  $X \supseteq Y$ , atunci  $X \rightarrow Y$ , arata ca un set de atribute permit determinarea unui alt subset care ii apartine;
- (*Regula de argumentare*)  $[X \rightarrow Y] \vdash [XZ \rightarrow YZ]$ , arata ca adaugarea aceluiasi set de atribute in membrul stang si in cel drept al dependentei determina o alta dependenta valida;
- (*Regula de tranzitivitate*)  $[X \rightarrow Y, Y \rightarrow Z] \vdash [X \rightarrow Z]$
- (*Regula de decompozitie*)  $[X \rightarrow YZ] \vdash [X \rightarrow Y]$ , arata ca se poate renunta la atribute din membrul drept al dependentei.  
Aplicarea repetata a acestei reguli poate descompune o regula de tipul FD  $X \rightarrow \{A_1, A_2, \dots, A_n\}$ , intr-un set de dependente  $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$ ;
- (*Regula de reuniune*)  $[X \rightarrow Y, X \rightarrow Z] \vdash X \rightarrow YZ$ , lucreaza in opozitie cu regula de decompozitie;
- (*Regula de pseudotranzitivitate*)  $[X \rightarrow Y, WY \rightarrow Z] \vdash WX \rightarrow Z$ .

Simbolul  $\vdash$  este interpretat ca asertarea validitatii sau deductia logica facuta pe baza membrului stang

## Normalizarea in baze de date

- Procesul de normalizare a fost propus de Codd (1972) si urmareste executia asupra unei scheme relatie o unei serii de teste pentru a cerceta apartenenta la forma normala. Codd propune trei forme normale (3NF), cea mai buna definire fiind data mai tarziu de Boyce si Codd, fiind cunoscuta sub numele de forma normala Boyce-Codd.
- Normalizarea datelor poate fi privita ca un proces in timpul caruia schemele relatie nesatisfacatoare sunt descompuse prin impartirea atributelor in scheme relatie mai mici ce poseda proprietatile dorite.

Forma normala ofera proiectantului bazei de date:

- un schelat formal pentru analiza schemelor relatie bazat pe chei si dependenta functionala intre atribute;
- o serie de teste ce pot elimina scheme relatii individuale astfel incit baza de date relationala poate fi normalizata in orice grad. Cind un test nu este trecut relatia va fi descompusa in relatii ce trec teste de normalitate.

Daca o schema relatie are mai mult de o cheie, fiecare este numita o cheie candidata. Una din cheile candidate este arbitrar denumita si **cheie primara** a schemei relatie, celelalte fiind numite si chei secundare. **Fiecare schema relatie trebuie sa aiba o cheie primara.**

## 1. Forma normala primara (1NF)

Forma normala primara este considerata ca fiind parte a definitiei formale a unei relatii. Ea nu permite atribute cu mai multe valori, atribute compuse sau combinatii ale lor. Aceasta stabeeste ca domeniul atributelor trebuie sa includa numai valori atomice si valoarea oricarui atribut intr-un n-uplu este o valoare unica in domeniul atributului respectiv. Cu alte cuvinte 1NF nu permite relatii in relatii sau relatii ca atribute ale n-uplurilor.

Considerind schema relatiei *DEPARTAMENT* in care cheia primara este D\_Nr si presupunind atributul Loc\_Dep, daca fiecare departament poseda un numar de locatii, nu o locatie unica, nu se obtine o forma normala primara.

Pentru a normaliza o relatie 1NF o solutie este de a descompune relativa in doua relatii, in cazul de fata *DEPARTAMENT* si *LOC\_DEP*. Ideea porneste de la eliminarea atributului *LOC\_DEP* din relatiile *DEPARTAMENT* si plasarea sa intr-o relatie separata alaturi de cheia primara *DEP\_NR* a relatiei *DEPARTAMENT*. O alta solutie pentru normalizare este cea prin care se pastreaza cate un n-uplu in relatiile *DEPARTAMENT* pentru fiecare locatie a departamentului. In acest caz obtinem o mare redondanta. Prima solutie este mult mai avantajoasa deoarece elimina redondantele.

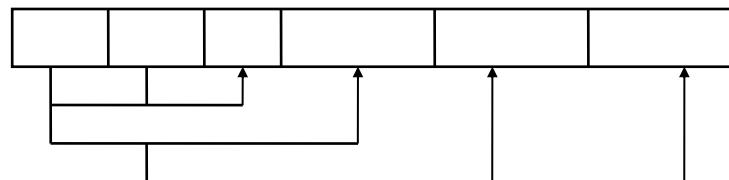
## 2. Forma normala secundara (2NF)

Forma normala secundara se bazeaza pe conceptul dependentei functionale complete. O *dependenta functionala*  $X \rightarrow Y$  este completa daca eliminarea oricarui atribut A din X distrugе dependenta functionala (oricare ar fi atributul A apartinind lui X,  $(X-(A)) \rightarrow Y$ ). O *dependenta functionala*  $X \rightarrow Y$  este *partiala* daca pot fi eliminate atribute din X fara ca dependenta functionala sa fie afectata (exista cel putin un atribut A apartinind lui X,  $(X-(A)) \rightarrow Y$ ). Ca exemplu, pentru relatia ANGAJAT-PROIECT specificata anterior {Ssn, P\_Nr}  $\rightarrow$  Ore este o dependenta functionala completa intrucit Ssn- $\rightarrow$ Ore si nici P\_Nr- $\rightarrow$ Ore. In schimb dependenta {Ssn,P\_Nr}  $\rightarrow$  A\_Nume este partiala intrucit Ssn  $\rightarrow$  A\_Nume.

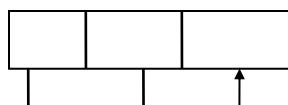
Daca o relatie nu contine numai dependente functionale complete ea se normalizeaza prin descompunere in mai multe relatii ce ofera functional dependenta completa, asa cum de ilustreaza in urmatorul exemplu.

Ex:

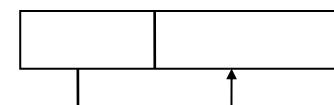
Ssn    P\_Nr    Ore    A\_Nume    P\_Nume    P\_Loc



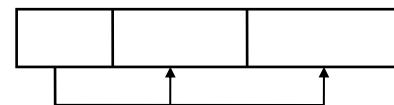
Ssn    P\_Nr    Ore



Ssn    A\_Nume



P\_Nr    P\_Nume    P\_Loc



S-a ilustrat descompunerea relatiei angajat\_proiect in trei relatii ce respecta forma normala de ordin trei (2NF)

### 3. Forma normala de ordin trei (3NF)

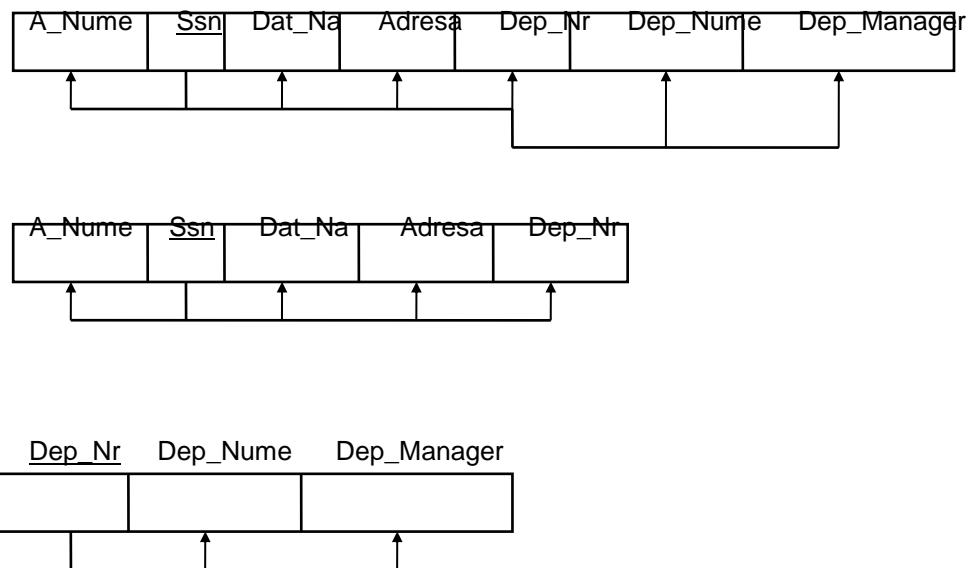
Forma normala de ordin trei este bazata pe conceptul dependentei tranzitive. O dependenta functionala  $X \rightarrow Y$  in relatia R este **tranzitiva** daca exista un set de atribute Z ce nu sunt un subset al oricarei chei din R, pentru care exista ambele dependinte:  $X \rightarrow Z$  si  $Z \rightarrow Y$ . De exemplu, dependenta  $Ssn \rightarrow Dep\_Manager$  este o dependenta tranzitiva in relatia ANGAJAT\_DEPARTAMENT.

Structura relatiei:

Angajat\_departament(A\_nume, Ssn, Dat\_na, Adresa, Dep\_nr,  
Dep\_nume, Dep\_manager)

Se poate spune ca dependenta atributului Dep\_Manager de atributul cheie Ssn este tranzitiva prin Dep\_Nr din cauza ca ambele dependente  $Ssn \rightarrow Dep\_Nr$  si  $Dep\_Nr \rightarrow Dep\_Manager$  sunt valide si Dep\_Nr nu este un subset al cheii relatiei ANGAJAT\_DEPARTAMENT. Intuitiv se observa ca dependenta atributului Dep\_Manager de Dep\_Nr este nedorita intrucit Dep\_Nr nu este o cheie in relatie.

Ex:



S-a descompus relatia pentru asigurarea formei normale de ordin 3.

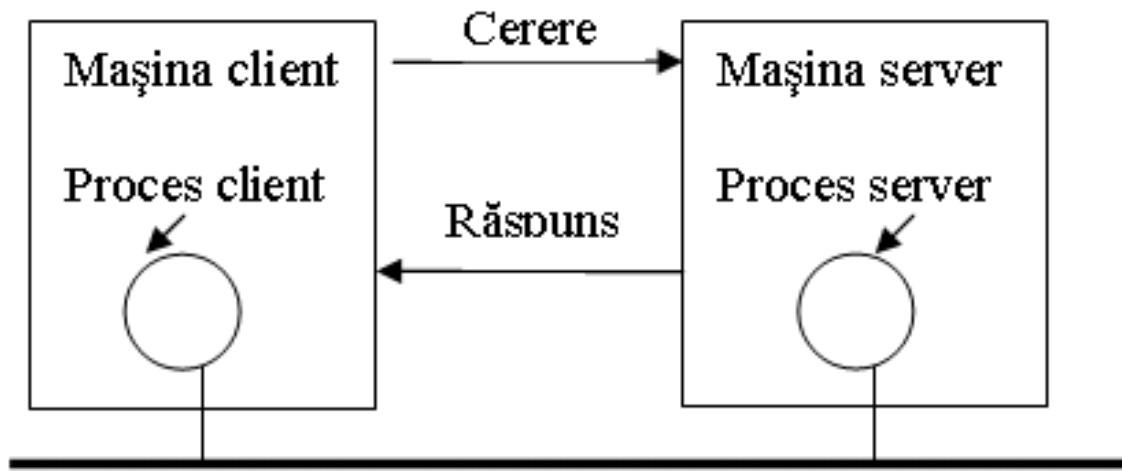
## Definitii generale 2NF si 3NF, Boyce-Codd (BCNF)

- O schema relatie R este 2NF daca fiecare atribut A nonprim in R nu este partial dependent de orice cheie din R.
- O schema relatie R este 3NF daca in orice dependenta functionala  $X \rightarrow A$  din R, este indeplinita una din conditiile: fie X este o supercheie a lui R, fie A este un prim atribut in R.
- Forma normala Boyce-Codd este o forma stricta 3NF, intrelegind prin aceasta ca fiecare relatie BCNF este in acelasi timp o relatie 3NF, cu toate ca o relatie 3NF nu este in mod necesar si o relatie BCNF.
- Dupa Boyce-Codd o schema relatie R este o forma normala daca pentru orice dependenta functionala  $X \rightarrow A$  in R, X este o supercheie in R. Diferenta intre BCNF si 3NF este marcata prin faptul ca 3NF permite ca A sa fie nonprim daca X nu este o supercheie. Afirmatia de mai sus justifica de ce o relatie BCNF este in acelasi timp si 3NF.

## Modelul Client-Server

- Ne propunem mai intai să identificăm principalele utilizări ale rețelelor de calculatoare. La nivelul organizațiilor ce dispun de un număr mare de calculatoare situate la distanță unul de altul, acestea nu trebuie să lucreze independent ci este nevoie de comunicare pentru ca la cerere să se poată corela informația stocată pe fiecare dintre ele.
- Un alt obiectiv este asigurarea unei mari flexibilități prin accesul la mai multe spații de stocare. Este curentă practica de a stoca aceleași informații pe mai multe mașini, astfel că la defecțiuni pot fi accesate datele alternative păstrate în copie pe altă mașină.
- La nivelul instituției se poate gândi o soluție prin care se asigură un singur calculator de mare viteză, având preț ridicat și mai multe calculatoare personale, cu posibilități modeste de prelucrare, pentru fiecare utilizator.
- O astfel de structură în care există mai multe servere de fișiere și utilizatori denumiți generic clienți formează modelul cunoscut sub denumirea de **model client – server**.

## Modelul Client-Server



Obs:

- În modelul client server procesul client trimite o cerere către procesul server, cerere ce implică realizarea unei anumite acțiuni.
- Serverul execută cererea formulată de către client și transmite rezultatul execuției acesteia clientului.
- Un server deservește, de regulă, un număr mare de clienți.

- O arhitectură client server nu necesită în mod obligatoriu o arhitectură de rețea, conceptul fiind perfect valabil și la implementările intrasistem, implementări ce nu duc la creșterea capacitatei de prelucrare.
- Ca urmare au apărut rețelele publice ce facilitează o serie de servicii cum sunt accesul la informație la distanță, schimbul de informație între persoane, divertisment interactiv etc.
- Accesul la informație la distanță poate lua forme multiple, de la simplul acces la instituțiile financiare pentru plăți electronice la cumpărăturile on-line.
- Informația este prezentată în diverse forme, de la informație de tip text la elemente multimedia: [imagine](#), [sunet](#), [film](#).
- Un canal uzual de comunicație în secolul XXI este poșta electronică sau e-mail, dar și o tendință de schimbare a tehnologiei pentru comunicația telefonică bazată pe [voice over IP](#).
- În domeniul comunicării între oameni sunt puse la punct tehnologii de comunicare audio și video pentru întâlniri virtuale numite și [videoconferințe](#).

## Concepțele securității și protecției datelor

- **securitatea datelor** - totalitatea masurilor de protecție împotriva accesului, distrugerii accidentale sau intentionate, a modificării neautorizate sau a divulgării acestora;
- **caracterul secret** este un concept ce se aplică la un individ sau organizatie și constă în dreptul acestora de a decide ce informații se pot folosi în comun și în ce condiții. Faptul că anumite date au caracter secret se stabilește de regula în afara sistemului de calcul;
- **confidențialitatea** se aplică la date și se referă la statutul acordat, aceasta reprezentând nivelul sau gradul de protecție ce trebuie asigurat informației respective;
- **integritatea** se referă la restricția de a nu schimba sensul datelor, adică sensul datelor să nu difere fata de cel inscris pe documentul sursă, impunind totodata ca datele să nu fie alterate accidental sau voit.

- Toate mediile de baze de date au incluse componente specializate prin care se restrictioneaza accesul la informatii.
- Protectia perimetrlui (specifica retelelor) poate asigura si protectia surselor de date.
- Dar baza de date nu poate fi ascunsa in spatele unui firewall.
- Securizarea unei baze de date nu inseamna numai stabilirea unei politici puternice de parolare ci si control adevarat al accesului.
- Sunt aspecte particulare dependente de sistemul de gestiune a bazei de date, multe dintre metodele de atac sunt comune, altele sunt particulare unui anumit mediu
- Chiar si la Oracle multe baze de date sunt configurate intr-un mod in care pot fi sparte relativ usor. Acest lucru este echivalent cu a spune ca Oracle nu este suficient de sigur, ci doar ca cele mai bune proceduri de protectie nu au fost luate in consideratie.
- Numarul de baze de date Oracle compromise nu se compara cu numarul de servere web atacate si compromise..

Ratiuni obiective ce justifica acest lucru:

- Sunt mult mai putine baze de date Oracle decat servere web;
- Cunostintele de securitate Oracle sunt limitate;
- Obtinerea unei versiuni de Oracle este dificila;
- Oracle ramane traditional in spatele unui firewall.
- Cu toate acestea factorii s-au schimbat mult in ultimii ani, mai ales prin cresterea interesului heackerilor pentru baze de date, descarcarea unui software Bazele de date au devenit mai simple, motiv pentru trebuie acordat un interes sporit securitatii

# Riscuri in securitatea BD, vulnerabilitati

- *Erorile software* – din aceasta categorie fac parte buffer overflows si ale erori de programare care duc la comenzi eronate ce altfel nu ar fi permise.
- *Arhitectura defectuoasa* – este rezultatul unei securitati neadecvate la proiectarea modului in care aplicatia lucreaza.
- *Configuratia eronata* – alegerea unei configuratii care compromite securitatea, multi dintre parametrii de securitate au valori implicite ce asigura securitate slaba. Exemplu pentru Oracle:  
REMOTE\_OS\_AUTHENTICATION care setat implicit True si permite utilizatorilor neautentificati sa se conecteze la baza de date;
- *Utilizarea incorecta* – se refera la constructia programelor utilizand ustensile de dezvoltare care pot fi utilizate intr-un mod in care blocheaza functionarea sistemului. Un astfel de exemplu este cel cunoscut sub numele de SQL Injection.

# Exemplu, Oracle Listener Service

Securitatea Oracle: *Listener Service*, un proxy care initializeaza conexiunea intre client si baza de date. Clientul directioneaza conexiunea la listener si acesta gestioneaza conexiunea la baza de date. Sistem de autentificare separata controlat si administrat din afara bazei de date. Sunt cateva ratiuni pentru care separarea dintre listener si securitatea bazei de date constituie probleme de securitate:

- Multi DBA nu realizeaza ca o parola trebuie setata la listener service. Aceasta este o facilitate mai putin documentata si nu este cunoscuta de catre toti administratorii de baze de date.
- Setarea de parola pentru listener service nu este simpla. Parola poate fi setata manual in fisierul de configurare listener.ora. Parola poate fi pastrata in clar sau rezultatul unei functii hash in fisierul de configurare. Daca se pastreaza hash setarea manuala nu poate fi facuta, pe cand la textul in clar oricine are acces sa citeasca directorul \$ORACLE\_HOME/network/admin va putea citi parola.

# Buffer overflow

- O alta problema cu listener service este scurgerea de informatii, problema identificata de James Abendschan. Daca se creaza un pachet cu o valoare incorrecta a campului ‘marime pachet’, listener va returna date in “buffer comanda” pana la marimea bufferului trimis. Daca o comanda trimisa de un utilizator are 200 de caractere si o alta comanda de 10 caractere cu o marime incorrecta este transmisa primele 10 caractere din comanda de 200 vor fi inlocuite si va returna comanda impreuna cu ultimele 190 caractere al comenzii utilizatorului anterior.
- Utilizand aceeasi tehnica poate fi trimis un pachet de dimensiune mare la listener, iar daca pachetul depaseste 1kb date, functionarea listener este compromisa. Prin umplerea stivei, un atacator poate determina executia unui cod arbitrar prin manipularea mecanismului SEH (Structured Exception Handling).

# SQL Injection

- Chiar daca baza de date este in spatele unui firewall nu inseamna ca nu trebuie sa ne facem griji ca aceasta ar putea fi atacata.
- Sunt o multime de alte forme de atac ce pot fi facute chiar cu existenta unui firewall. Cea mai obisnuita metoda utilizata in zilele noastre este SQL Injection.
- SQL Injection nu este un atac direct la baza de date, ci este cauzat de modul in care aplicatiile web sunt dezvoltate.
- Daca atacatorul introduce o simpla ghilimea in campul formei web si introduce o alta interogare, face posibila si executia celei de a doua cereri.
- Cel mai simplu mod de verificare a existentei unei vulnerabilitati este cel de includere a unei ghilimele in fiecare camp al formei si verificarea rezultatului.
- Unele dintre site-uri vor returna un cod de eroare, reclamand o eroare de sintaxa, altele nu raspund nimic.

# SQL Injection

- Desigur ca acestea sunt inca vulnerabile dar sunt mai greu de exploatat intrucat nu furnizeaza raspuns la mesajele de eroare. Acest tip de atac se manifesta la toate mediile de baze de date, chiar daca modul de lucru difera de la o baza la alta. SQL Injection se bazeaza pe incercarea atacatorului de a modifica o interogare ca in exemplul de mai jos.

```
Select * from my_table where column_x = '1'
```

in :

```
Select * from my_table where column_x = '1' UNION  
select password from DBA_USERS where 'q'='q'
```

In acest exemplu o singura cerere este convertita in doua cereri. Sunt de asemenea modalitati de modificare a criteriilor WHERE pentru actualizarea sau stergerea randurilor la care initial nu de doreste acest lucru. La unele medii de baze de date se permite includerea unei a doua comenzi in cerere, daca nu atacatorul poate suplimenta sfarsitul cererii. In cazul nostru 'q'='q' la sfarsit este utilizata deoarece va face ca aceasta clauza sa fie evaluata ca adevarata.