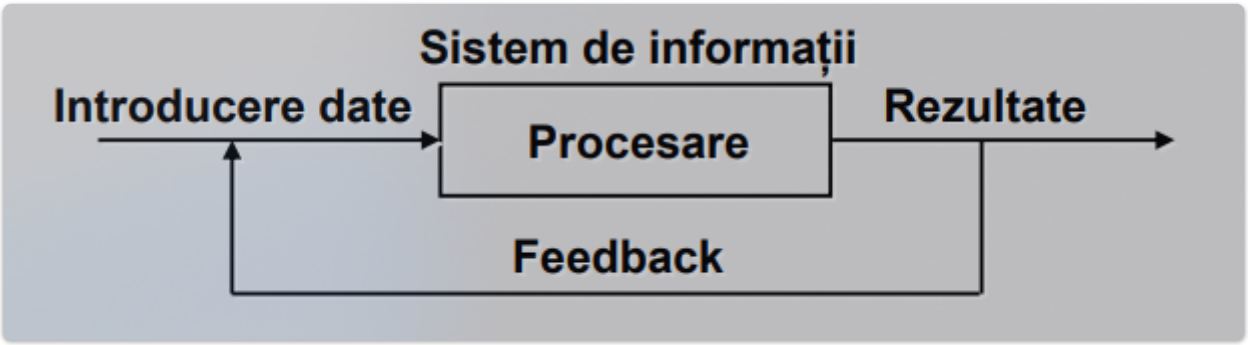
#PD

Procesarea Datelor

Se referă la aplicarea unui set de operaţii specifice pe un grup de date sau într-o bază de date.
Procesarea datelor se efectuează cu ajutorul sistemelor de informaţii
Un sistem de informaţii este un concept larg care cuprinde atât sistemele de calcul cât şi dispozitivele asociate acestora



Dispozitive folosite la introducerea datelor:

- Tastatura
- Scaner
- Microfon
- Camera video

Procesarea datelor presupune transformarea datelor de intrare in rezultate utile prin intermediul unor operatii cum ar fi:

- Efectuarea de comparatii
- Efectuarea de actiuni alternative
- Stocarea datelor pentru utilizari ulterioare

Rezultatele se prezinta sub forma de rapoarte si documente

Feedback-ul se foloseste pentru a efectua modificari ale datelor introduse si a fazei de procesare din cadrul sistemului de informatii

Faza de procesare permite:

- Detinerea controlului coplet asupra datelor
- Obtinerea de informatii pe baza datelor stocate

Cele mai folosite operatii intalnite pe parcursul procesarii datelor sunt:

- Gruparea datelor pe categorii

- Agregarea datelor
- Determinarea procentelor
- Crearea de tabele

Scopul activitatilor de procesare a datelor este acela de a transforma o cantitate uriasa de date in informatii pe baza carora sa se poata:

- Lua decizii
- Stabilii stategii
- Indeplini functii de conducere

Informatia

Notiune foarte veche, cu grad mare de complexitate si generalitate

Semnificatii:

- Unitate de masura si stiinta calculatoarelor
- Suport al cunostintelor intr-un anumit domeniu
- Stire
- Noutate
- Etc.

Sensul general acceptat:

- > *"informaţia defineşte fiecare dintre elementele noi conţinute în semnificaţia unui simbol sau grup de simboluri, într-o comunicare, ştire, semnal, grup de imagini etc. prin care se desemnează concomitent o situaţie, o stare, o acţiune etc."\ - DEX*

Caracteristici:

1. Precizia- Informatiile precise sunt lipsite de erori
2. Completitudinea - Informatiile complete contin toate datele importante
3. Eficienta - Informatiile trebuie sa fie usor de obtinut
4. Flexibilitatea - Informatiile trebuie sa fie permita folosirea in mai multe scopuri, nu doar unul
5. Fiabilitatea - Informatiile fiabile sunt informatii credibile
6. Relevanta - Informatiile relevante sunt informatii importante pentru luarea unei decizii
7. Simplicitatea - Informatiile trebuie sa fie usor de gasit si inteles
8. Disponibilitatea - Informatiile trebuie sa fie puse la dispozitia utilizatorului atunci cand sunt cerute in orice moment
9. Veridicitatea - Informatiile trebuie sa fie verificata pentru a obtine asigurarea ca sunt precise

Data

Pentru a putea fi percepută informaţia trebuie exprimată într-o formă concretă care să poată fi analizată, manipulată şi transformată, iar această formă se materializează în conceptul de dată

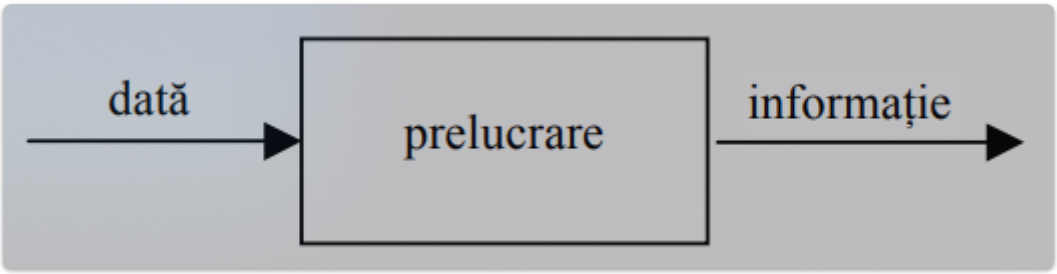
Definitie:

Un număr, o mărime, o relaţie etc. care serveşte la rezolvarea unei probleme sau care este obţinută în urma unei cercetări urmând a fi supusă unor prelucrări

Relatia dintre date si informatii:

Datele sunt utilizate pentru:
- transmiterea informatiilor
- pastrarea informatiilor
- obtinerea de noi informatii prin prelucrari

Numai prin asociere cu realitatea, se poate spune ca se prelucreaza informatii



Măsura cantităţii de informaţie

Măsura cantităţii de informaţie este dată de inversul logaritmului în baza 2 din probabilitatea de apariţie a unui eveniment

Preocupările pentru măsurarea cantităţii de informaţie provin din teoria statistică a comunicaţiei şi se bazează pe ideea că informaţia exprimă incertitudinea înlăturată prin realizarea unui eveniment dintr-un set de evenimente posibile

Etalonul pentru măsurarea cantităţii de informaţie trebuie să fie în aşa fel ales încât să poată măsura informaţiile, să le poată compara, indiferent de modul cum sunt emise, transmise sau recepţionate

Exemplu:

Se dispune de o urna cu x bile albe si y bile negre

La extragerea unei bile se obtine o anumită informaţie deoarece se elimină incertitudinea bila albă-bila neagră (dacă urna ar conţine doar bile negre sau doar bile albe, atunci s-ar cunoaşte rezultatul: bila neagră respectiv bila albă)

Fie

- X- un experiment
- $x_1, x_2, \ldots, x_n$ - un numar finit de evenimente elementare independente (rezultatul experimentului)
- $p_1, p_2, \ldots, p_n$ - probabilitatile de realizare a evenimentului
- Se consideră că experimentul X este un sistem complet de evenimente, adică prin efectuarea sa se obţine cu siguranţă unul din rezultatele $x_k \in X$
- Inseamna ca: $0 \leq P(x_k) \leq 1$, pentru orice $k \in [1, n]$, $\Sigma P(x_k) = 1$
- Experimentul poate fi descris sub forma:

$$\begin{vmatrix} x_1 & x_2 & \ldots x_k & \ldots x_n \\ p_1 & p_2 & \ldots p_k & \ldots p_n \end{vmatrix}$$

- Inainte de realizare, experimentul X conţine un anumit grad de nedeterminare care va fi înlăturat prin realizarea sa deoarece se obţine unul din cele "n" evenimente posibile
- Realizarea experimentului furnizează o anumită cantitate de informaţie, care înlocuieşte nedeterminarea iniţială
- Informaţia obţinută este cu atât mai mare cu cât nedeterminarea înlăturată este mai mare
- Ca urmare, se poate folosi pentru cantitatea de informaţie aceeaşi unitate de măsură utilizată în cazul aflării gradului de nedeterminare
- Variabilitatea datelor sau cantitatea de informaţie furnizată de către un experiment în urma căruia se obţine un rezultat particular este exprimată cu ajutorul entropiei
- Pentru o variabilă aleatoare Z, distribuită după legea $\phi(z)$, entropia este:
  - $H = -\int \phi(z) \log \phi(z)$
- Nedeterminarea unui experiment X este o funcţie de probabilitate de realizare a evenimentelor componente, egală cu cantitatea medie de informaţie furnizată de realizarea unui eveniment, adică:
  - $H(X) = H(p_1, p_2, \ldots, p_n)$
- In 1948, Claude Shannon, matematician şi inginer american stabileşte pentru această funcţie expresia:
  - $H(X) = H(p_1, p_2, \ldots, p_n) = -\Sigma_{k=1}^{n} p_k \log_2 p_k$
- Formula reprezintă măsura unei mărimi, dar nu rezolvă problema unităţii de măsura care să exprime această cantitate
- Care ia, de fapt, în calcul entropia care măsoar soară împr împrăştierea
- Shannon a continuat cercetările unui precursor în domeniul teoriei informaţiei, R.V. Hartley, care încă din 1928 a introdus noţiunea de cantitate de informaţie
- Shannon a propus ca unitatea de măsura a cantităţii de informaţie să fie informaţia generată de realizarea unui experiment cu două evenimente având probabilităţi egale de realizare
- Aceasta unitate de măăsura poart sura poartăă numele de BIT (binary digit)
- Dacă se aplica modelul anterior pentru un experiment X, cu două evenimente echiprobabile se obţine:

    - $X = \begin{vmatrix} 0 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{vmatrix}$
- Ceea ce duce la:
    - $H(X) = -\Sigma_{k=1}^{n} p_k \log_2 p_k = -(\frac{1}{2}\log \frac{1}{2} + \frac{1}{2}\log_2 \frac{1}{2}) = -(-\frac{1}{2} - \frac{1}{2}) = 1$

Transformarea datelor in informatie si invers

Uneori, în practică, termenul de informaţie este utilizat şi pentru a desemna datele sau invers, dar acest lucru este impropriu

Datele se refera la numere, fapte, diferite documente etc

Informaţiile se referă la date organizate, date care au fost filtrate şi ordonate după anumite criterii

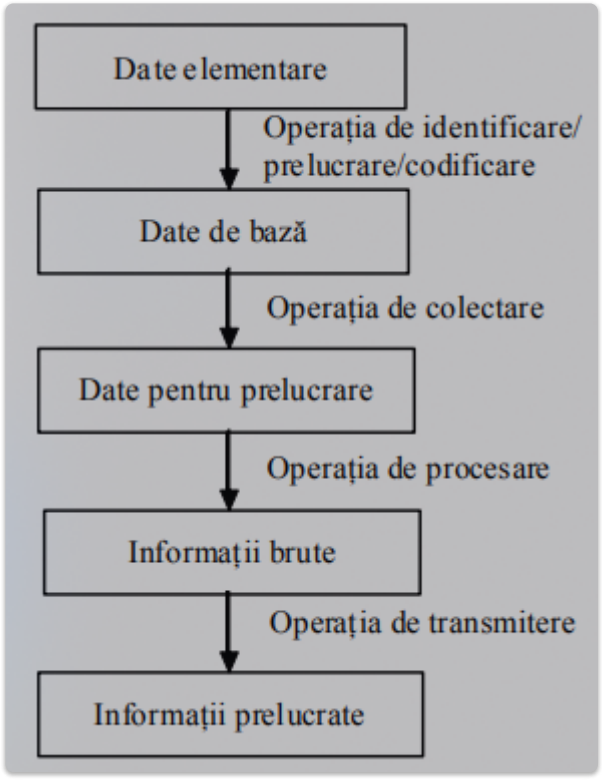
- **Informaţile pot fi privite sub trei aspecte:**
    - sintactic, ca mod de reprezentare prin numere, mărimi, sunete etc.
    - semantic, din punct de vedere al sensului (semnificaţiei) pentru cel ce o recepţionează
    - pragmatic, adică din punct de vedere al utilităţii

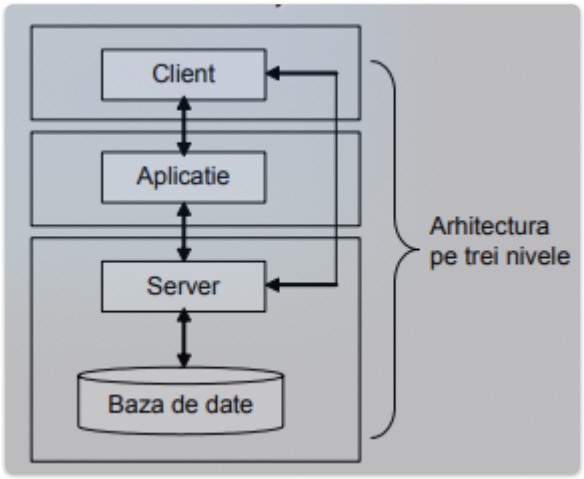
In general, în cadrul unei activităţi, este necesară prelucrarea datelor în vederea obţinerii informaţiilor necesare luării deciziilor, datorită următoarelor motive:
1. informaţia apare de cele mai multe ori în alt loc decât cel în care este utilizată;
2. informaţia apare de cele mai multe ori în alt moment decât cel în care este utilizată, ceea ce impune păstrarea informaţiilor.
3. informaţia apare de cele mai multe ori sub o formă diferită de cea în care este utilizată.

Prelucrarea informaţiei are ca obiect adaptarea informaţiilor elementare pentru a fi direct utilizabile

Principalele etape de prelucrare a datelor pot fi prezentate în următoarea succesiune:

- Operaţia de identificare/prelucrare/codificare realizează codificarea datelor elementare, reţinându-le cu scopul conservării, transformării şi comunicării lor
- Operaţia de colectare presupune regruparea datelor elementare necesare prelucrării şi se poate realiza în 2 moduri:
    - in timp real
    - decalat in timp
        - – prima modalitate este proprie consultării imediate a datelor prezente permanent în sistem
        - – în cazul celei de-a doua modalităţi setul de date de aceeaşi natura este constituit şi conservat într-un fişier
- Operaţia de procesare reprezintă transformarea datelor destinate prelucrării în informaţii pe baza unui algoritm

- Operaţia de transmitere oferă informaţii prelucrate la locul, momentul şi sub forma dorită
  - 

- Gradul de utilizare a informaţiei şi eficienţa sa în diversele activităţi umane depind de indicii de calitate specifică:
  - precizia (cantitatea de informaţii corecte în raport cu volumul total de informaţii)
  - oportunitatea sau actualitatea (utilitatea informaţiei într-un anumit interval de timp)
  - completitudinea (informaţii cât mai complete despre un anumit fenomen)
- Cea mai eficientă soluţie o reprezentă înmagazinarea datelor în baze de date, urmată de utilizarea acestora în cadrul diverselor aplicaţii
  - 

- Clientul poate fi o persoan – clientul poate fi o persoanăă, o func , o funcţţie sau un serviciu
- Decizia ce trebuie luată pentru rezolvarea unei anumite sarcini se ia de către client, care este responsabil de toate acţiunile şi deciziile sale
- Datele sistemului reprezintă o sursă comună, la dispoziţia tuturor, impunându-se totuşi anumite restricţii accesării acestora, din motive de securitate
- Structura datelor este rigidă, rămânând aceeaşi pe întreaga desfăşurare a activităţii, cu excepţia cazului în care aceasta este modificată de către administrator

- Se constată faptul că principala operaţie utilizată în acest gen de aplicaţii este aceea de regăsire a datelor în scopul obţinerii de informaţii (obiectivul principal al oricărei baze de date)
- Unei baze de date i se pun întrebări pentru a afla ceea ce interesează la un moment dat
- In afară de operaţia de căutare (principală), în baza de date mai au loc şi altfel de operaţii necesare întreţinerii bazei de date, cum ar fi:
  - adaugarea de date noi
  - stergerea datelor care si-au pierdut importanta sau actualitatea
  - modificarea datelor pentru a fi in pas cu realitatea
- **Metafora bibliotecii si a bibliotecarului**
  - bazele de date reprezinta de fapt, replica informatizata a bibliotecii traditionale
  - bibliotecarul a capatat un nume nou, pe masura cresterii competentelor (administrator al bazei de date)
  - **Critica**
    - bibliotecile lucrează cu informaţii în timp ce bazele de date lucrează cu date
    - administratorul bazei de date are un rol pe care bibliotecarul nu îl are: acela de a transforma datele în informaţii şi pe acestea înapoi în date, sursă de apariţie a majorităţii erorilor întâlnite în bazele de date
    - o bază de date poate fi privită ca un model al unei realităţi definite prin intermediul datelor
    - modelarea se face cu ajutorul unor obiecte ce prezintă diverse caracteristici şi între care se stabilesc o serie de asocieri ce descriu în amănunt modul în care obiectele (caracteristicile lor) contribuie la o descriere cât mai exactă a realităţii
    - datele în sine, nu au nici un fel de semnificaţie
    - a obţine informaţie înseamnă, de fapt, a introduce datele disponibile întrun a numit context conferindu-le în acest fel o anume semnificaţie
    - transformarea datelor în informaţie este, însă, de multe ori, extrem de dificilă datorită complexităţii realităţii care trebuie modelată şi reprezintă cea mai importantă sursă de erori care contribuie la obţinerea în final a unor informaţii care se dovedesc a nu fi cele aşteptate
    - una dintre cauzele majore de apariţie a defectelor în bazele de date o reprezintă crearea unei scheme defectuoase a bazei de date

Prelucrarea datelor

- **Operaţiile de prelucrare a datelor, pot fi, în principal, grupate pe două mari componente:**
  - componenta de înmagazinare a datelor în cadrul sistemului reprezentată, de obicei, de baze de date manipulate cu ajutorul unor mecanisme proprii împreună cu care formează aşa-numitele sisteme de gestiune a bazelor de date;
  - componenta de transfer a datelor între aplicaţii care îndeplineşte rolurile de conectare la bazele de date şi de gestiune a datelor din sistem, reprezentată de diverse tehnologii cum ar fi COM, DCOM, ODBC, ADO şi, mai nou, XML.

- **Probleme:**
  1. aplicaţiile utilizate astăzi sunt mari şi complexe, cer mari resurse de timp din partea producătorilor, dificultăţi şi costuri ridicate de întreţinere, apariţia de riscuri crescute la defectare în cazul adăugării unor componente noi
  2. aplicaţiile sunt compacte, dispun de multe caracteristici extrem de utile, dar marea majoritate a acestor caracteristici nu poate fi modificată, eliminată sau înlocuită cu versiuni mai noi
  3. aplicaţiile nu sunt uşor de integrat (datele şi proprietăţile unei aplicaţii nu sunt disponibile decât cu dificultate altor aplicaţii, chiar dacă aplicaţiile sunt scrise în acelaşi limbaj de programare şi rulează pe aceeaşi maşină)
  4. sistemele de operare prezintă şi ele o serie de inconveniente. Ele nu sunt suficient de bine modularizate, fiind dificil de înlocuit, modificat sau actualizat serviciile oferite de acestea într-o modalitate simplă şi flexibilă
  5. modelele utilizate la programare, din diferite motive, nu sunt consistente. Chiar şi atunci când aplicaţiile prezintă facilităţi de cooperare, serviciile lor sunt disponibile într-o modalitate diferită de cea oferită de sistemul de operare. Modelele de programare sunt foarte diverse în cazul în care serviciul rulează în acelaşi proces cu clientul, într-un proces separat pe aceeaşi maşină, sau pe o maşină separată în cadrul unei reţele de calculatoare
- **Strategii**
  1. un set generic de facilităţi necesare găsirii şi utilizării unui anumit serviciu (indiferent dacă acesta este oferit de aplicaţie, de sistemul de operare sau de ambele), de utilizare a proprietăţilor oferite de un anumit serviciu, de extindere şi dezvoltare a unei noi versiuni fără ca aceasta să împiedice utilizatorii vechii versiuni să o mai poată folosi
  2. utilizarea conceptelor programării orientate pe obiecte atât în cadrul sistemelor cât şi în cadrul aplicaţiilor pentru a putea folosi în programare noua generaţie de instrumente orientate pe obiecte (în sprijinul unei gestiuni mai eficiente a programelor ce devin din ce în ce mai complexe) care să ajute la creşterea modularităţii, a reutilizării codurilor şi la facilitarea dezvoltării de alte proiecte cu cât mai puţine componente noi
  3. utilizarea arhitecturii pe trei nivele pentru a putea beneficia de comunicarea cu diverse dispozitive, cu serverele aflate în reţea sau cu alte sisteme disponibile într-o modalitate cât mai sigură cu putinţă
  4. utilizarea mediilor distribuite pentru a oferi utilizatorilor şi aplicaţiilor o singură vedere asupra sistemului şi a permite folosirea serviciilor în cadrul unei reţele de calculatoare indiferent de locaţie, arhitectură a maşinii sau mediu de implementare
- **Solutii:**
  1. programarea orientatăă pe obiecte
     - oferă o mare putere o mare putere şi o flexibilitate ridicată
     - permite crearea unor seturi de obiecte ce pot fi reutilizate în cadrul altor aplicaţii

- nu există un cadru elaborat de standarde prin care obiectele create de diferiţi producători să poată comunica între ele în cadrul aceluiaşi proces şi cu atât mai puţin în procese separate sau la distanţă

2. crearea de componente reutilizabile
   - o componentă este o parte reutilizabilă de cod în format binar ce poate fi introdusă în cadrul altor componente ale diverşilor producători cu relativă uşurinţă
   - Exemplu:
     - un analizator de sintaxă de la un producător ce poate rula în cadrul diverselor procesoare de texte ce provin de la alţi producători
   - componentele software trebuie să adere la standardul binar pentru a putea fi folosite în exterior
   - implementarea lor internă nu este supusă nici unui fel de constrângeri
   - componentele software pot fi construite folosind fie limbaje procedurale, fie limbaje orientate pe obiecte

Concepte:

Date

- Fapte culese din lumea reala pe baza de observatii si masuratori
- Constituie orice mesaj primit de un receptor sub o anumita forma
- **Caracteristici** :
  - Insusiri, proprietati
  - Determina modul de organizare a datelor
  - Permit extragerea esentei intelesului datelor

Model de date

- O multime formala si consistenta de reguli
- Trebuie alese şi folosite acele entităţi, acţiuni, precum şi relaţiile dintre ele care prezintă interes pentru utilizator

Colectie de date

- Un ansamblu de date organizat după anumite criterii

Multime

- O colecţie de obiecte care au identitate proprie şi sunt caracterizate de o condiţie de apartenenţă

Organizarea datelor

- Definesc un aspect sau o proprietate

- Determina modul de organizare a datelor
- Permit extragerea esentei intelesului datelor
- Descrierea mulţimii obiectelor reale sau abstracte ale unui domeniu se realizează cu ajutorul unei serii de caracteristici/atribute, sub forma: $C_i, i = 1, n$

Caracteristici / Atribute

- Fiecărei caracteristici i se asociază o mulţime de valori numite date X_i
- Pentru a arăta că o dată aparţine sau nu mulţimii de valori ai caracteristicii C_i se scrie: $x_i \in X_i$
- Exemplu:
  - Nume student=$nume_1, nume_2, \ldots, nume_n$, in care:
    - *Nume student* reprezinta caracteristica
    - $Nume_i$ reprezinta datele

Familie de caracteristici

- Reprezintă mulţimea de caracteristici necesare pentru descrierea unei mulţimi: $\Lambda = C_i | i \in N$
- Exemplu
  - $\Lambda = Nume, prenume, sex, situatie, adresa$
  - Fiecare element al multimii reprezinta o caracterisitca ce ajuta la descrierea unei persoane

Relatii

- Au semnificatia de raport, legatura, asociere
- **Definitie:**
  - Fie Λ_1 si Λ_2 multimi; se numeşte relaţie între cele două mulţimi un triplet $v = \Lambda_1, G, \Lambda_2$, unde G este o submultime a produsului cartezian $\Lambda_1 \times \Lambda_2$
  - Multimea G se numeste graful corespondentei
  - Fiind data o familie $\Lambda = C_i | i \in N$. Intre caracteristicile acesteia se pot stabili o mulţime de tipuri de relaţii: *binare, ternare, ...n-are, de echivalenţă, de ordine, de apartenenţă*
- Proprietati matematice
  - reflexivitate $\forall x, (x, x) \in R$, adica xRx
  - simetrie $\forall x, (y, x) \in R$, adica $xRy \Leftrightarrow yRx$
  - tranzitivitate $\forall x, (y, x) \in R$, adica $xRy \wedge yRz \Rightarrow xRz$
  - antisimetrie $(x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$
- Relatia poate fi interpretata ca fiind un tabel care este alcatuit din coloane si randuri
- **O coloana** se numeste camp sau atribut (caracteristica)
- **Un rand** se numeste inregistrare (tuplu)

Proprietati

- Nu sunt admise tupluri duplicat;
- Tuplurile sunt neordonate;
- Atributele sunt neordonate;
- Toate valorile atributelor sunt atomice (nu se mai pot descompune)

Relatia binara

- Se poate stabili între valorile a două caracteristici C_1 şi C_2 ale aceleiaşi familii sau familii diferite
- Este o submultime a produsului cartezian $X_1 \times X_2$ ce satisface o anumita relatie R in care:
  - $X_1 =$ domeniul valorilor caracteristicii C_1
  - $X_2 =$ domeniul valorilor caracteristicii C_2
- Pentru a arata ca elementele $x_{1i} \in x_1$ si $x_{2k} \in x_2$ ele sunt asociate prin relatia R sub forma $x_{1i}Rx_{2k}$

Relatia de ordin n

- Reprezintă o parte a produsului cartezian dintre valorile caracteristicilor unei familii
- Determinarea unei părţi de produs cartezian se poate realiza în două moduri:
  1. Prin enumerarea elementelor produsului cartezian ce fac parte din relaţie
  2. Prin utilizarea unui predicat P care să realizeze selectarea produsului cartezian;
- **Exemple:**
  - Intr-un tabel referitor la FACULTATE "profesorii ce au aceeaşi dată de angajare"
  - In mulţimea notelor obţinute de studenţi "note ce au aceeaşi valoare"
- O relaţie de echivalenţă în cadrul unui fişier sau 2 înregistrări asociate prin aceeaşi relaţie se numesc **echivalenţe**
- Mai multe înregistrări echivalente formează o **clasă de echivalenţă**
- Reprezintă o submulţime cu o anumită semnificaţie, care prezintă componentele:
  1. O familie de caracteristici alcătuită din atribute ce definesc aspecte ale obiectelor din lumea reală;
  2. Un predicat aplicat familiei de caracteristici ce conduce la o submulţime ce defineşte o relaţie de ordine între caracteristici;
  3. O suită temporală $T=\{t_0, t_1, ...t_j,...\}$ ce defineşte un decalaj al timpului în intervale discrete;
  4. Posibilitatea modificării în orice moment t_j a unei relaţii asociată predicatului
- Descrierea datelor se întâlneşte sub denumirile de catalog de sistem, dicţionar de date sau meta-date (date despre date)

- **Exemplu:**

  - O colecţie de date ce reprezintă o submulţime a studenţilor dintr-o facultate
  - Predicatul P poate consta din enumerarea caracteristicilor colecţiei

CodS	Nume	Prenume
001	Banu	Andrei
002	Manta	Andrei
003	Dima	Cristina
004	Stroie	Camelia
005	Radu	Tiberiu
006	Dima	Carmen

Colectie de date

- Colectia de date se regaseste sub denumirile:
  1. Fişier, în cazul organizării clasice
  2. Entitate (domeniu), în concepţia bazelor de date în reţea
  3. Tabel, relaţie, vedere, cluster, în concepţia bazelor de date relaţionale
- Sistemele de gestiune a bazelor de date relationale (SGBDR) folosesc limbajul standard de manipulare a bazelor de date SQL (Structured Query Language) în una din versiunile sale
- Se pune la dispoziţie o singură frază cu structura generală minimă

```
SELECT - lista coloanelor ce apar in raspuns
FROM - tabelele din care se preiau coloanele
[WHERE] - conditiile ce trebuie indeplinite (predicat)
```

Operatii pe multimi folosite in varianta SQL

- **Reuniunea**
  - Reprezinta multimea tutror tuplurilor ce apartin fie unei relatii fie alteia
    - $$R_1 = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}, R_2 = \begin{pmatrix} a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \\ a_5 & b_5 & c_5 \end{pmatrix}$$
  - Drept rezultat apare o nou – drept rezultat apare o nouă rela relaţie
    -

$$R_3 = R_1 \cup R_2 = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \\ a_5 & b_5 & c_5 \end{pmatrix}$$

- Din punct de vedere al standardului SQL acest lucru se poate scrie sub forma:

```
SELECT R1.* FROM R1
UNION
SELECT R2.* FROM R2
```

Intersectia

- Reprezita forma tuturor tuplurilor ce aparţin atât lui R_1 cât şi lui R_2

$$R_1 = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}, R_2 = \begin{pmatrix} a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \end{pmatrix}, R_3 = R_1 \cap R_2 = \begin{pmatrix} a_2 & b_2 & c_2 \end{pmatrix}$$

- Din punct de vedere al standardului SQL acest lucru se poate scrie sub forma

```
SELECT R1.* FROM R1
INTERSECT
SELECT R2.* FROM R2
```

- Sau

```
SELECT NUME_TABEL1.CAMP1.CAMP2….CAMPN FROM NUME_TABEL1
INTERSECT
SELECT NUME_TABEL2.CAMP1.CAMP2…CAMPN FROM NUME_TABEL2
```

- **Observatii**
  - SQL Server, dar şi alte sisteme de gestiune a bazelor de date nu cunosc noţiunea de *INTERSECT*, pentru astfel de situaţii aplicându-se operaţia de joncţiune.

Diferenta

- Reprezintă mulţimea tuturor tuplurilor ce aparţin unei relaţii, dar nu aparţin şi celeilalte relaţii:
- Fie

$$R_1 = \begin{pmatrix} a_1 & b_1 & c_1 \end{pmatrix}, R_2 = \begin{pmatrix} a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \\ a_5 & b_5 & c_5 \end{pmatrix}$$

- Diferenta:

$$R_3 = R_2 - R_1 = \begin{pmatrix} a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix}$$

- Ceea ce in SQL se scrie sub forma:

```
SELECT * FROM R1
    WHERE (a,b,c) NOT IN (SELECT * FROM R2)
```

Produsul cartezian

- Fie

$$R_1 = \begin{bmatrix} a \\ b \end{bmatrix}, R_2 = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Produsul cartezian:

$$R_1 \times R_2 = \begin{bmatrix} ax \\ ay \\ az \\ bx \\ by \\ bz \end{bmatrix}$$

- Semnificaţia practică a produsului cartezian corespunde generării unei relaţii din două relaţii prin combinarea fiecărui tuplu din prima relaţie cu fiecare tuplu al celei de-a II-a relaţii
- Ceea ce in SQL se scrie sub forma

```
SELECT * FROM R1, R2
```

- **Observatii:**
  - Produsul cartezian este foarte rar folosit in practica
  - Operatorii neexperimentaţi cad de multe ori în capcana întinsă de o cunoaştere insuficientă şi de o abordare necorespunzătoare din punct de vedere conceptual

Operatia de selectie

- Operatorul algebric de selecţie produce o subrelaţie sau subset al unei relaţii date supuse operaţiei de selecţie
- Subrelaţia va conţine multitudinea tuplurilor relaţiei supuse selecţiei care satisfac un predicat specificat
- Predicatul poate fi o expresie logica

```
SELECT R1.* FROM R1 WHERE predicat
```

- ## Operatia de protectie
  - Proiecţia unei relaţii determină o altă relaţie obţinută prin selectarea atributelor specificate şi eliminarea tuplurilor duplicat (folosirea clauzei *DISTINCT*)
  - **Exemplu**
    - Se dă un tabel referitor la studenti
    - Se cere să se realizeze proiecţia structurii tabelului pe atributul *Nume*
    - Ca rezultat se va obţine mulţimea numelor studenţilor, fiecare nume luat o singură dată
    - In limbaj SQL aceasta se traduce prin:

```
SELECT DISTINCT NUME FROM STUDENT;
```

- ## Operatia de jonctiune
  - Este o derivaţie a produsului cartezian şi este în anumite situaţii identică cu acesta
  - Presupune utilizarea unui calificator care să permită compararea valorilor diferitelor sau aceloraşi câmpuri din 2 relaţii sau dintr-o singură relaţie
  - Are sens atunci când la intersectarea relaţiilor nu apare mulţimea vidă, adică există cel puţin un atribut comun
  - Nu este obligatoriu ca denumirea relatiilor sa fie identica
    $JOIN(R_1, R_2|\theta(R_1 \times R_2[\theta]))$, in care
    - θ este calificator multiatribut ce permite compararea atributelor relatiei R_1 cu atributele relatiei R_2
  - Presupunem înmulţirea fiecărui tuplu (înregistrare) dintr-o relaţie R_1 cu fiecare tuplu dintr-o relaţie R_2 ce îndeplineşte o anumită condiţie
  - Pentru realizarea operaţiei de joncţiune, cele 2 relaţii trebuie să aibă un atribut comun, adică: $R_1 \cap R_2 \geq 0$
  - Atributul comun trebuie să aibă semnificaţie de cheie externă în relaţia R_1 şi cheie primară în relaţia R_2
  - Ca rezultat se va obţine o nouă tabelă ce va conţine doar tuplurile ce îndeplinesc condiţia prealabilă

- **Exemplu**
  Student

CodS	Nume	Prenume
001	Banu	Andrei
002	Manta	Andrei
003	Dima	Cristina

CodS	Nume	Prenume
004	Stroie	Camelia
005	Radu	Tiberiu
006	Dima	Carmen

↓

Note

CodS	Curs
005	Fizica
005	Chimie
002	Fizica
002	Chimie
005	Istorie
005	Engleza
006	T.B.D.
006	P.C.

In limbaj SQL:

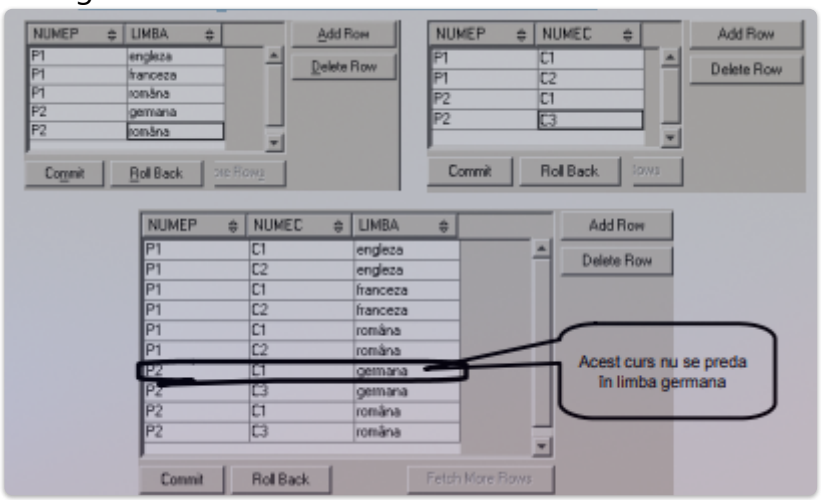
```
SELECT STUDENT.CODS, NUME, PRENUME, CURS
    FROM STUDENT INNER JOIN NOTE ON STUDENT.CODS = NOTE.CODS
```

ceea ce este echivalent cu:

```
SELECT STUDENT.CODS, NUME, PRENUME, CURS
    FROM STUDENT, NOTE
    WHERE STUDENT.CODS = NOTE.CODS
```

- ## Erori aparute la conversia date-informatie
  - Pentru evitarea redundantelor se apeleaza la asa numita normalizare a bazelor de date
  - Pentru evitarea introducerii de mai multe ori a aceleiasi inregistrari se apeleaza la cheia primara
  - Cheia primara este singurul mecanism ce poate fi folosit cu scopul de a identifica in mod unic o inregistrare
  - Intr-un tabel nu poate exista decât o singură cheie primară, dar aceasta poate fi alcătuită din două sau mai multe coloane ale tabelului (sursă de erori - interogarea

nu poate folosi asocierea prin intermediul cheii primare, deoarece aceasta nu are corespondent în tabelul asociat)

- **Exemplu:**
  - Intr-un sistem de gestiune al informaţiilor se păstrează date despre o facultăţi
  - Printre altele, există şi date referitoare la cadrele didactice care sunt titulare ale unor discipline
  - Se mai oferă date şi despre limbile străine cunoscute de către cadrele didactice respective
  - Se doreşte să se afle în ce limbi străine sunt predate cursurile în cadrul instituţiei de învăţământ respective
  - Dacă schema bazei de date ar fi incorect concepută, s-ar putea obţine rezultatele din figură
  - 

  - Tabelele care conţin datele sunt tblPLimba care are cheia primară alcătuită din coloanele NumeP şi Limba, respectiv tblPCurs care are cheia primară alcătuită din coloanele NumeP şi NumeC
  - In urma interogarii tabelelor tblPLimba si tblPCurs se obtin date false

```
SELECT DISTINCT TBLPLIMBA.NUMEP, TBLPCURS.NUMEC, TBLPLIMBA.LIMBA
    FROM TBLPLIMBA INNER JOIN TBLPCURS ON TBLPLIMBA.NUMEP=TBLPCURS.NUMEP
```

- Din punctul de vedere al sistemului de gestiune al bazelor de date lucrurile sunt corecte
  - Rezultatul putea fi anticipat şi din punct de vedere matematic, deoarece avem de a face aici cu un produs cartezian
  - Din punct de vedere al logicii aplicaţiei lucrurile nu sunt în ordine atâta timp cât sunt obţinute informaţii false:
  - un profesor predă un curs într-o limbă nerepartizată cursului respectiv
  - **Observatii**
    - Utilizarea de informaţii false conduce la formularea unor concluzii care pot contribui la luarea unor decizii ce pot avea efecte dezastruoase pentru utilizator (“mai bine nu am date decât să folosesc informaţii false”)

Jonctiunea

Utilizare

Tabelul de jonctiune se foloseste atunci cand lucram cu relatii de tipul multe-la-multe.

Exemplu

Presupunem ca lucram cu doua tabele *Student* si *Clase*. FIecare student poate apartine mai multor clase sau nici uneia. De asemenea, fiecare clasa poate avea multipli studenti sau nici unul.

O junctiune ne permite crearea relatiilor de tip multi-la-multi si, mai important, impiedica adaugarea intratilor duplicate

Tabelele *Student* si *Clase* arata astfel:

```
CREATE TABLE Student
(
    IdStudent INT PRIMARY KEY,
    NumeStudent VARCHAR(50) NOT NULL
)
CREATE TABLE Clase
(
    IdClase INT PRIMARY KEY,
    NumarClasa INT NOT NULL
)
```

Acum ca am creat cele doua tabele va trebui sa formam tabelul de junctiune ce le va lega intre ele.
Tabelul de junctiune este creat utilizand cheile primare pentru tabelele *Student* si *Clase*

```
CREATE TABLE ClaseStudent
(
    IdStudent INT NOT NULL,
    IdClase INT NOT NULL,
    CONSTRAINT PK_ClaseStudent PRIMARY KEY
    (
        IdStudent,
        IdClase
    ),
    FOREIGN KEY (IdStudent) REFERENCES Student (IdStudent),
    FOREIGN KEY (IdClase) REFERENCES Clase (IdClase)
)
```

Am creat un tabel cu coloane pentru *IdStudent* si *IdClase*. Acest tabel foloseste, de asemenea, o combinatie dintre aceste doua coloane ca si cheie primara. Inseamna ca fiecare pereche Clasa-Student este unica. Fiecare student poate apartine mai multor clase, fiecare clasa poate apartine mai multor studenti , dar fiecare pereche poate aparea o singura data.

Coloanele din acest tabel de jonctiune sunt configurate ca si chei secundate pentru tabelele *Student* si *Clase*. Lucrul acesta este important deoarece nu ne permite sa adaugam studenti pentru o clasa care nu exista sau sa stergem o clasa din baza de date daca inca sunt studenti care ii apartin

Pentru a vedea ce student apatrin carei clase putem sa folosim tabelul de junctiune si urmatoarea interogare:

```
SELECT NumeStudent, NumarClasa
FROM ClaseStudent
JOIN Student ON Student.IdStudent = ClasaStudent.IdStudent
JOIN Clase ON Clase.IdClase = ClasaStudent.IdClase
```

Theta si Echi-Jonctiunea

- SQL nu prezinta clauze sau operatori speciali pentru jonctiune
- O jonctiune este vazuta ca o combinatie dintre produsul cartezian si selectie
- Theta-Jonctiunea a doua relatii R_1 si R_2 se scrie:

```
SELECT *
    FROM R1,R2
    WHERE R1.A >=R2.E
```

Sau

```
SELECT *
    FROM R1,R2
    WHERE RI.A=R2.E
```

Sinonime:

In frazele *SELECT* tabelelor li se pot asocia nume mai scurte:

```
SELECT Nume, Nota
    FROM Student S INNER JOIN Note N ON S.CodS=N.CodS
```

Exista siutatii in care este obligatoriu sa se foloseasca sinonimele, cum ar fi cele in care se efectueaza o jonctiune a tabelei cu ea insasi

Exemplu:

```
SELECT S2.CodS
    FROM Student S1 INNER JOIN Student S2 ON S1.Data_inceput=S2.Data_inceput
```

Jonctiunea externa

RIGHT OUTER JOIN

Exemplu:

```
SELECT *
    FROM R1 RIGHT OUTER JOIN R2 ON R1.C=R2.C
```

FULL OUTER JOIN

Specifica faptul ca o inregistrare ce apartine fie relatiei din stanga, fie relatiei din dreapta ce indeplineste conditia se include in setul de rezultate, iar coloanele corespunzatoare din celalalta relatie sunt setate pe valoareal NULL

Exemplu:

```
SELECT *
    FROM R1 FULL OUTER JOIN R2 ON R1.C=R2.C
```

Concepte

Baza de date

- Contine toate datele necesare despre obiectele de interes, relatiile logice intre aceste date si tehnicile de prelucrare corespunzatoare
- In bazele de date are loc o integrare a datelor (mai multe fisiere sunt privite in ansamblu, eliminandu-se pe cat posibil datele redundante)
- Se permite accesul simultan la aceleaşi date, situate în acelaşi loc sau distribuite spaţial, a mai multor persoane
- **Exemplu:**
  - Pentru o facultate, pot fi păstrate, de exemplu, pe perioade mari de timp, date referitoare la studenţi, personal, săli, planuri de învăţământ, aparatură şi alte elemente despre care diverse persoane pot cere informaţii la un moment dat
  - Intre aceste elemente există tot felul de relaţii cum ar fi: unii studenţi urmează anumite cursuri, unele cursuri au loc în anumite săli, unele aparate se află în anumite săli, unele persoane pot ţine anumite cursuri ş.a.m.d

Sistem de gestiune a bazelor de date (SGBD)

- Permite utilizatorului sa aiba acces de date, pentru a obţine informaţii, prin folosirea unui limbaj de nivel înalt, apropiat de modul obişnuit de exprimare
- Utilizatorul face abstracţie de algoritmii aplicaţi privind selecţionarea datelor implicate şi a modului lor de memorare
- SGBD-ul poate fi privit ca o interfaţă între utilizatori şi sistemul de operare

Componente de baza

- Limbajul de definire a datelor (LDD) sau DDL (Data Definition Language), care descrie :
  - Structura bazei de date;
  - Componentele bazei de date;
  - Relatiile dintre componentele bazei de date;
  - Drepturile de acces ale utilizatorilor la baza de date;
  - Restrictiile in reprezentarea datelor;
  - Etc.
- Limbajul de prelucrare a datelor (LPD) sau DML (Data Manipulation Language), ce permite operaţii asupra datelor aflate în baza de date, cum ar fi:
  - Inmagazinarea datelor în baza de date;
  - Inserarea unui element nou în baza de date;
  - Eliminarea unui element din baza de date;
  - Modificarea unui element;
  - Căutarea elementelor în bazele de date;
  - Realizarea diverselor statistici asupra datelor
  - Etc.
- Limbajele DDL şi DML se introduc sub forma unor extensii în cadrul altor limbaje de programare numite limbaje gazdă
- Calităţile SGBD-ului depind în mare măsură de calităţile limbajului gazdă utilizat
- Foloseşte conceptele matematice de algebră relaţională pentru a grupa datele în mulţimi şi a stabili relaţii între submulţimile (domeniile) comune
- Fundamentat teoretic de căătre E. F. Codd
- In cadrul modelului relaţional, datele sunt separate în mulţimi care prezintă asemănări cu structura unui tabel
- Tabelele sunt alcatuite din coloane (campuri) si inregistrari (tupluri)

Exemplu:

- Un tabel din cadrul unei baze de date relaţionale ce conţine datele personale ale studenţilor unei facultăţi poate să înceapă cu următoarele date: nume, curs, nota
- Aceste trei tipuri de date formează câmpurile din tabelul următor :

Nume	Curs	Nota
Student1	Curs1	9.00
Student2	Curs1	10.00
Student3	Curs1	4.00

Limbajul SQL

- Este un limbaj utilizat la crearea structurilor de baze de date precum şi la manipularea datelor în cadrul acestor structuri
- Exemple de acţiuni ce se pot realiza prin intermediul limbajului SQL :
  - Creare tabele
  - Adaugare date
  - Stergere date
  - Combinare date
  - Declansare de actiuni in functie de modificarrile aduse bazei de date
  - Memorarea interogarilor in cadrul programului sau bazei de date
- **Caracteristici SQL-2**
  - Jonctiunea externa (OUTER JOIN);
  - Raportare standardizata a erorilor;
  - Dictionarul de date (metadatele);
  - posibilităţi de modificare a bazei de date prin introducerea cauzelor DROP, ALTER, GRANT, REVOKE;
  - Modificări şi ştergeri referenţiale în cascadă;
  - Niveluri de consistenţă a tranzacţiilor
- **Caracteristici SQL-2006**
  - Modalităţi de import şi stocare a datelor XML în cadrul bazelor de date SQL
  - Manipularea datelor XML în cadrul bazelor de date
  - Prezentarea atât a datelor XML cât şi a celor SQL în format XML
  - Facilităţi ce permit aplicaţiilor să integreze în codul SQL structuri ale limbajelor XQuery sau XML Query în scopul accesului concurenţial atât la documentele XML cât şi la datele SQL
- Cea mai folosită instrucţiune din SQL este *SELECT* (utilizată pentru regăsirea datelor în bazele de date
- ANSI SQL recomandă un nucleu format dintr-un grup de funcţii dedicate manipulării datelor
- Sistemele de baze de date trebuie să posede, în mod obligatoriu, instrumente pentru păstrarea integrităţii datelor şi asigurarea securităţii acestora
- Primii trei mari producători de sisteme de gestiune profesionale a bazelor de date, care deţin peste 90% din piaţa mondială, sunt ORACLE, IBM şi Microsoft ce promovează produsele :
  - Oracle
  - DB2
  - SQL Server

Istoric

- Momentul decisiv în naşterea SQL îl constituie lansarea proiectului System/R de către firma IBM în 1974

- Comitetul pentru baze de date X3H2, arondat Institutului Naţional American pentru Standarde (ANSI) a finalizat în 1986 standardul SQL-86 (X3 reprezintă secţiunea care se ocupă de sisteme de prelucrare a informaţiilor)
- Organizaţia Internaţională pentru Standarde (ISO) a adoptat primul document, aproape identic cu ANSI SQL-86, pe care l-a publicat în 1987 sub numele de ISO 9075-1987 Database Language SQL.
- In 1989 apare SQL-89 care mai este denumit SQL-1 – ANSI X3.135- 1989, respectiv ISO 9075:1989
- Pentru a umple golurile SQL-1, ANSI şi ISO au elaborat în 1992 versiunea SQL-2, ANSI X3.135-1992 (Database Language SQL), respectiv ISO/IEC 9075:1992
- SQL-3 a fost publicatăă în cea mai mare parte în iulie 1999 în cea mai mare parte în iulie 1999
- Odată cu publicarea acestui standard, SQL iese din sfera relaţionalului şi a normalizării relaţiilor şi se îndreaptă spre lucrul cu obiecte, introducând noţiuni noi în vederea gestionării obiectelor complexe şi persistente, cum ar fi:
  - Generalizare şi specializare;
  - Moşteniri multiple;
  - Polimorfism;
  - Incapsulare;
  - Tipuri de date definite de utilizator;
  - Suport pentru sisteme bazate pe gestiunea cunoştinţelor;
  - Expresii privind interogări recursive şi instrumente adecvate de administrare a datelor
- SQL-3 introduce şi alte noţiuni nelegate strict de lucrul cu obiecte, cum ar fi:
  - Declanşatoare;
  - Proceduri stocate;
  - Seturi de caractere naţionale;
  - Tabele virtuale actualizabile;
  - Roluri pentru definirea pofilelor de securitate
- Versiunea apărută în anul 2003 introduce caracteristicile legate de folosirea tehnologiei XML, secvenţe standardizate şi coloane cu valori autogenerabile (proprietatea Identity în SQL Server)
- In anul 2006 a apărut cea mai recentă versiune a standardului SQL care are un echivalent în standardul ISO/IEC 9075-14:2006 şi în care se definesc modurile în care limbajul SQL poate fi folosit împreună cu tehnologia XML

Crearea unui proiect de baza de date

Schema

- Numele obiectelor bazei de date impreuna cu prorpietatile lor si asocierile dintre ele
- Se construieste pe baza unui Model Conceptual

- **Exemple de instrumente ajutatoare folosite la elaborare**

  - Diagrame entitate-relatie (ER)
  - Diagrame Unified Modeling Language (UMF)

- Ambele folosesc aceleasi concepte de entitate, atribute si relatii dintre entitati

Diagrama UML

- **Avantaje**

  - Principalele reguli care se folosesc la modelarea situatiei reale sunt formulate sub forma unor proprietati ale constructiilor folosite in cadrul modelului
  - Documenteaza cerintele informationale intr-un format clar si precis
  - Abordarea grafica a modelarii o face usor de inteles
  - Prin simplificarea realitatii modelul este mai usor de utilizat
  - Modelul poate fi imbunatatit prin reinginerie (inginerie inversata) sau prin crearea mai multor diagrame
  - La realizarea modelului se iau in considerare doar aspecte esentiale, ce prezinta importanta, ignorandu-se cele minore

- **Dezavantaje**

  - Orice simplificare introduce o serie de imperfectiuni
  - Nu orice se poate reprezenta grafic
  - Nu exista un model unic
  - Au caracter general, fiind destinate in special modelarii in cazul programarii orientate pe obiecte (modelul relational are caracteristici specifice, deosebite fata de modelul orientat pe obiecte)
  - Crearea se bazeaza pe informatiile documentate in specificatiile si cerintele utilizatorului

- **Conversia diagramei UML in schema instantei**

  - Schema instantei este compusa din randuri care definesc caracteristicile critice ale bazei de date
  - Entitatile din diagrama UML devin tabele, atributele devin coloane ale acestor tabele, iar asocierile devin chei externe

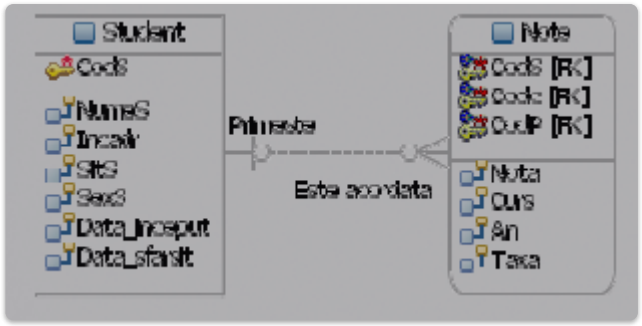
- **Exemple:**

  - Trebuie cunoscute notele acordate studentilor
  - Trebuie cunoscute numele si numarul matricol are fiecarui student
  - Un student va avea doar o singura nota la un curs

Numele coloanei	CodS	Curs	Nota	An	Taxa
Tipul cheii	PK	PK	PK		
Valori NULL	NN	NN	NN		
Cheie externa	CodS				
Tipul de data	char	char	dec	char	int
Lungime max.	5	15	4,2	1	7
Exemplu	005	Fizica	6	1	350000

Diagrama Informa

Exemplu:

- Un fragment al modelului conceptual obtinut pe baza unei diagrame simple a relatiilor dintre tipurile de entitati luate in considerare la reprezentarea unie facultati
- 

Crearea modelului fizic

Crearea tabelelor

- Este folosita instructiunea *CREATE TABLE*
- Se realizeaza prin definirea numelui tabelului, precum si a numelor coloanelor tabelului si a descrierii acestora folosind o instructiune SQL

  Sintaxa:

```
CREATE TABLE <nume_tabel>
(
<nume_camp> <tip_data> [[CONSTRAINT <nume_restrictie> [DEFAULT <valoare_impl
)
```

Tipuri de date (SQL-92)

- *SMALLINT*: intregi (4 pozitii, reprezentare pe 16 biti);
- *INTEGER* sau *INT*: intregi (9 pozitii, 32 biti);

- *NUMERIC* (p,s) sau *DECIMAL* (p,s) sau *DEC* (p,s): reale cu p pozitii, din care s la partea fractionara;
- *FLOAT*: reale, virgula mobila (20 pozitii pentru mantisa);
- *REAL*: real, virgula mobila (cu precizie mai mica decat FLOAT);
- *DOUBLE PRECISION*: reale, virgula mobila, dubla precizie (30 pozitii pentru mantisa);
- *CHAR(n)* sau *CHAR VARYING(n)* sau *CHARACTER VARYING(n)*: sir de caractere de lungime variabila (max. 254);
- *DATETIME*: data calendaristica;

Dictionar de date

- Dictionarul de date este cea mai importanta forma de documentatie pentru proiectantul de baze de date

Obiective

1. Descrierea scopului bazei de date si a utilizatorilor.
2. Realizarea documentatiei bazei de date. Aceasta poate insemna oricare dintre urmatoarele specificatii: pe ce sistem a fost creata; dimensiunea prestabilita a bazei de date sau dimensiunea fisierului jurnal.
3. Includerea codului sursa SQL pentru oricare fisier script de instalare sau dazinstalare a unei baze de date. Aceasta operatie include documentatia de utilizare a instrumentelor pentru import/export.
4. Asigurarea descrierii amanuntite a fiecariu tabel din baza de date si a scopurilor acestora in contextul utilizarii lor.
5. Documentarea structurii interne a fiecărui tabel. Aceasta poate include toate câmpurile şi tipurile de date, cu comentarii, toţi indecşii şi toate vederile.
6. Includerea codului sursă SQL, pentru toate procedurile rezidente şi pentru toţi declanşatorii.
7. Asigurarea unei descrieri a cerinţelor bazei de date, cum ar fi folosirea valorilor unice sau a valorilor NOT NULL. De asemenea, ar trebui menţionat dacă aceste constrângeri sunt forţate la nivelul SGBDR sau dacă programatorul bazei de date trebuie să verifice aceste constrângeri în cadrul codului sursă

Instrumente CASE

Tehnologia CASE (Computer Aided Software Engineering) este un domeniu de integrare si sinteza ce incorporeaza elementele din proiectarea asistata a calculator, ingineria programarii, proiectarea sistemelor informatice, baze de date si alte domenii ale informaticii.

Ajuta la usurarea muncii de realizare a produselor informatice

Denumire alternativa: IPSE - Integrated Project Support Enviroment

Facilitati

1. Suport pentru una sau mai multe metodologii de analiză şi proiectare a aplicaţiilor informatice, prin editoare de diagrame şi text;
2. Stocarea şi regăsirea datelor din dicţionarul de date (repository) prin utilitare specifice (browser);
3. Verificarea automată a consistenţei şi completitudinii datelor printr-un analizor ce conţine reguli specifice pentru fiecare metodologie;
4. Suport pentru realizarea de prototipuri prin limbaje de nivel înalt şi generatoare;
5. Suport pentru conducerea proiectului prin instrumente de generare a grafurilor (resurse, versiuni);
6. Generarea documentaţiei de realizare a sistemului;
7. Generarea automată a codului program, pornind de la specificaţiile de proiectare;
8. Tehnica de inginerie inversată (reverse engineering) prin care se permite revenirea dintr-o etapă de realizare a aplicaţiei la o etapă precedentă pentru eventuale modificări;
9. Adaptabilitatea şi extensibilitatea trebuie să fie proprietăţi de bază ale instrumentelor CASE

Clasificare

Criterii

Dupa aria de cuprindere a ciclului de realizare a aplicatiei:

- Upper CASE (front-end) oferă suport pentru primele etape de realizare a aplicaţiilor (planificarea realizării, analiza şi specificarea cerinţelor, proiectarea logică);
- Lower CASE (back-end) oferă suport pentru ultimele etape de realizare a aplicaţiilor (proiectarea fizică, elaborarea programelor, testarea, întreţinerea şi dezvoltarea sistemului)

Dupa dimensiunea suportului oferit:

- Instrumente CASE propriu-zise oferă suport pentru o singură activitate din cadrul unei etape de realizare a aplicaţiilor (editoare de diagrame şi text, dicţionarul de date, analiza specificaţiilor de sistem, generatoare de aplicaţii, depanatoare, conducerea proiectului). Aceste instrumente pot fi verticale (cele ce se pot folosi într-o singură etapă – generatoare, compilatoare etc.) sau orizontale (cele care se pot folosi în mai multe etape (elaborarea documentaţiei, editoare de diagrame etc.).
- Bancurile de lucru CASE (workbenches) oferă suport pentru o etapă din ciclul de realizare a aplicaţiei. Ele integrează un set de instrumente din categoria precedentă (instrumente pentru analiză, pentru proiectare, pentru programare, pentru implementare).
- Mediile CASE oferă suport pentru cea mai mare parte (sau toate) dintre etapele de realizare a aplicaţiei într-un singur produs informatic. Ele integrează instrumente CASE din categoriile precedente.

Dupa metodologia implementata:

- Instrumente CASE pentru modele structurate;
- Instrumente CASE pentru modelarea datelor – asistă proiectantul de baze de date la definirea entităţilor şi a legăturilor dintre ele (diagrame ER);
- Limbaje din generaţia a patra pentru baze de date (4GL); sunt limbaje de nivel înalt care oferă suport pentru manipularea datelor (SQL, generatoare);
- Instrumente CASE pentru metode orientate pe obiecte care sunt încă relativ puţine, dar care tind să se extindă datorită avantajelor oferite (flexibilitate sporită, reutilizarea proiectelor dezvoltate anterior etc.)

Arhitectura mediului CASE

- Este data de componentele care integreaza mai multe instrumente CASE
- Dictionarul central de date (repository) este componenta de vaza a unui mediu CASE. In el se stocheaza integrat si neredundant descrierea completă a sistemului: modulele sistemului şi legăturile deintre ele, structura bazei de date, machetele de intrare/ieşire, informaţii pentru conducerea proiectului.
- Editoarele pentru introducerea şi actualizarea datelor din sistem: pentru realizarea graficelor şi a schemelor, pentru reprezentarea tabelelor, pentru texte.
- Utilitare pentru translatarea datelor şi implementarea instrumentelor CASE în mediul integrat. Pentru fiecare instrument din mediul CASE datele au o anumită formă specifică. Mediul asigură utilizarea datelor pentru toate instrumentele prin conversii, precum şi integritatea şi coerenţa acestora.
- Generarea automată a codului se realizează pornind de la specificaţiile de proiectare conţinute în dicţionarul central de date. O mică parte din mediile CASE include această componentă deoarece ea necesită o descriere în detaliu a sistemului şi utilizarea unor tehnici complexe de structurare a sistemului.
- Facilităţi de inginerie inversată care permit ca pe baza descompunerii şi analizei codului de program şi a descrierii datelor să fie refăcute specificaţiile de proiectare şi să fie actualizate informaţiile din dicţionarul de date. Se poate astfel relua etapa precedentă din cea curentă.
- Facilităţile pentru conducerea proiectului permit colectarea şi utilizarea informaţiilor privind planificarea şi controlul realizării proiectului. Informaţiile sunt stocate în dicţionarul de date.

Instrumentele CASE pentru analiza structurata

Componenta

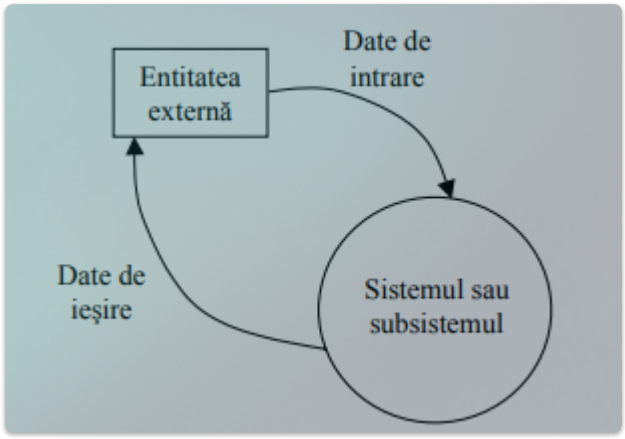
- Editoare grafice si de text pentru specificarea cerintelor (genereaza diagrame si analiza ale descrierii sistemului)
- Generatoare de documentatie in urma etapei de analiza si actualizarea ei

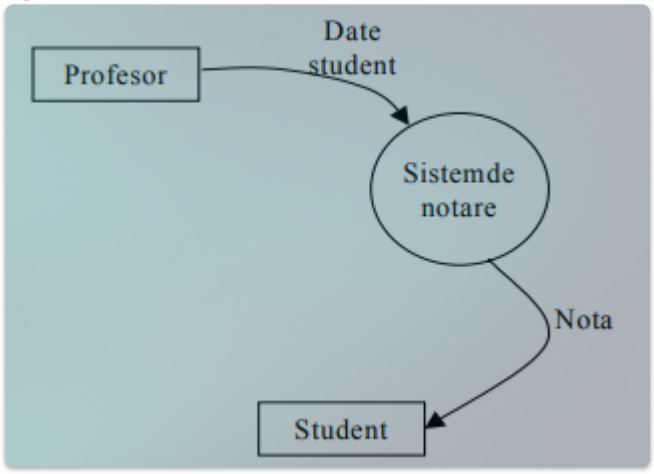
Exemple

Diagrame de flux a datelor (DFD)

- Ilustreaza modul de deplasarea a datelor, definind intrarile si iesirile sistemului, efectuand in acelasi timp si o delimitare a acestuia cu scopull miscorarii complexitatii domeniului de interes si a usurarii comunicarii in ceea ce priveste intelegerea unui anumit sistem sau a unei parti a acestuia
- Instrumentul CASE contine un editor de diagrame care construieste mai intai diagrama contextuala, dupa care fiecare proces din aceasta diagrama poate fi detaliat intr-o diagrama de nivel inferior
- Pentru fiecare proces care nu mai admite o descompunere ulterioara trebuie intocmita specificatia de proces
- In analiza se folosesc trei tipuri de diagrame (de continut, fizice si logice)

Diagrame DFD de continut

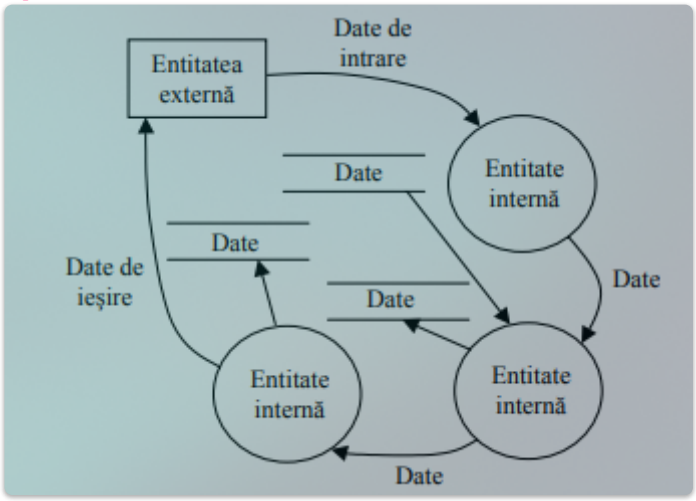
- Definesc procesele care au loc intre una sau mai multe entitati externe (reprezentate prin intermediul unor dreptunghiuri si care reprezinta sursele de date) si o singura entitate interna (reprezentata printr-un cerc) care reprezinta sistemul sau subsistemul aflat in discutie
- Liniile fluxului de date sunt indreptate de la sau spre entitetea externa
- In astfel de diagrame trebuie sa existe cel putin o intrare si cel putin o iesire
- 

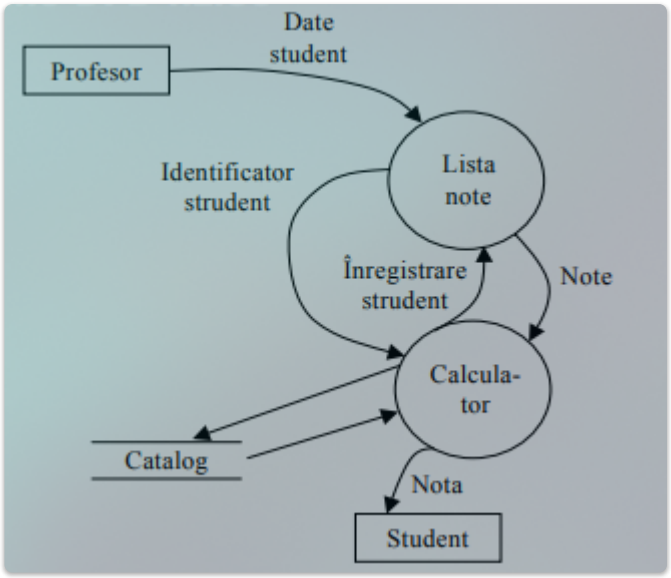
- **Exemplu**
  - 

Diagrame DFD fizice

- Definesc procesele care au loc intre una sau mai multe entitati externe (reprezentate prin intermediul unor dreptunghiuri si care reprezinta sursele de date) si mai multe entitati

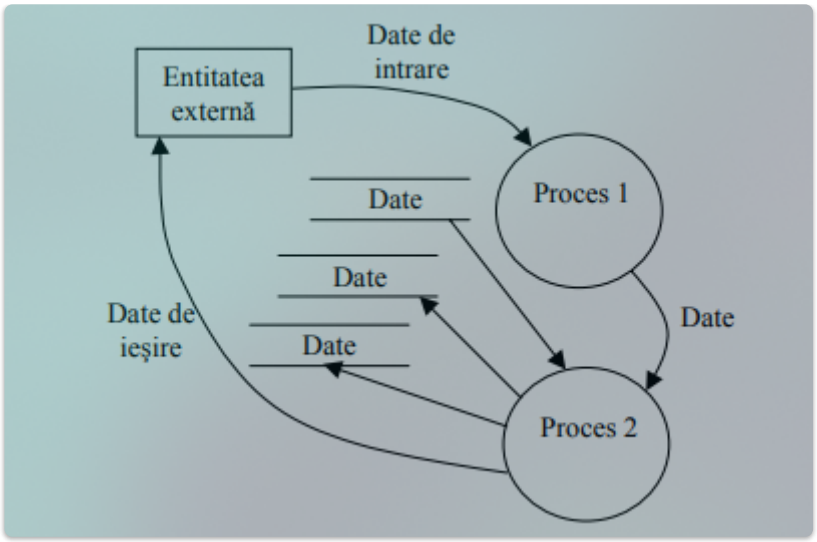
interne (reprezentate prin cercuri)

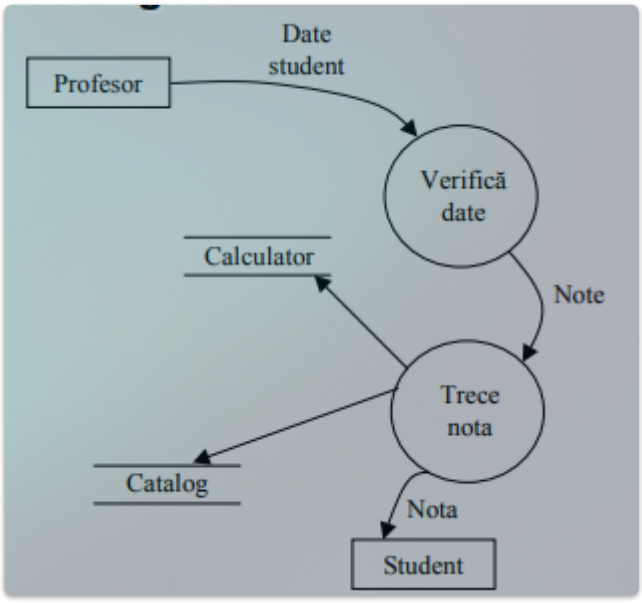
- Liniile fluxului de date provin de la sau sunt indreptate spre entitatea/entitatile externe si sunt indreptate spre sau provin de la entitatile interne
- Trebuie sa existe si alte elemente de stocare a datelor, cum ar fi de exemplu, fisierele, reprezentate prin doua linii paralele
- **Exemplu**
  - 

  - 

Diagrame DFD logice

- Stabilesc pasii care trebuie urmati pe parcursul defasurarii unui proces
- Liniile fluxului de date sunt indreptate de la sau spre entitatea/entitatile externe spre/de la procese
- In astfel de diagrame trebuie sa existe si alte elemente de stocare a datelor, cum ar fi de exemplu, fisierele, reprezentate prin doua linii paralele
- **Exemplu**





Specificatiile de proces (minispecificatii)

- Sunt descrieri algoritmice ale prelucrarilor realizate de un proces de nivel atomic
- Orice specificatie trebuie sa contina:
  - Numele
  - Lista intrarilor de date
  - Lista iesirilor de date
  - Corpul specificatiilor (descrierea algoiritmica)
- Tehnicile de specificare a proceselor sunt:
  - Scheme logice
  - Pseudocod
  - Tabele de decizie
  - Diagrame
  - Limbaje de programare

Dictionarul de date

- Contine descrierea tuturor elementelor ce apar in diagramele de flux (fluxurile de date, procesele, datele stocate)

- Ofera o definire riguroasa si detaliata a datelor din sistem

Medii CASE pentru analiza structurata

Exemplu

- Westmount I – Case Yourdon

Instrumente CASE pentru proiectarea structurata

Modelarea datelor

- Se realizeaza cu ajutorul unor instrumente de modelare, cele mai utilizate fiind diagramele entitate-relatie sau diagramele UML
- Se reprezinta grafic entitatile logice si legaturile dintre acestea (modelarea), rezultand o diagrama
- Porninid de la diagrama si utilizand descrierea colectiilor si a atrubutelor din dictionarul de date, instrumentele CASE pot genera automat schema bazei de date si codul necesar crearii structurii fizice a bazei de date
- **Exemplu**
  - IBM Rational Data Architect IBM Rational Data Architect

Puncte slabe

- Generarea automata a codului pornind de la specificatiile de programare
- Reutilizarea modulelor de program existente
- Integrarea diferitelor instrumente CASE intr-un mediu (lipsa de standarde)
- Dezvoltarea iterativa a aplicatiilor (inginerie inversata)
- Dictionarul de date contine doar definitiile datelor, daca ar trebui sa contina si structura de apel a programelor si algoritmi
- Codul sursa generat poate fi ulterior modificat manual, dar acest lucru poate duce la pierderea consistentei si poate sa nu mai perminta ingineria inversata (pentru rezolvarea problemei ar fi necesara o biblioteca de module utilizator)

Clauze SQL

In gramatica, o clauza este un cuvant sau un grup de cuvinte care in mod obisnuit este alcatuit dintr-un subiect si un predicat, fiind posibil ca, in anumite limbi/ limbaje sau in anumite clauze, subiectul sa nu apara in mod explicit.

Cea mai simpla propozitie este alcatuita dintr-o singura clauza, dar propozitiile complexe pot contine mau multe clauze.

Este posibil ca o clauza sa contina o alta clauza sau sub-clauza.

In limbajul SQL, s-a optat pentru folosirea conceptului de clauza si nu pentru folosirea celui de instructiune, deoarece se considera ca, de fapt, toate operatiile de regasire a datelor in bazele de date se pot reduce la niste cereri formulate in mod asemanator clauzelor.

Clauza principala folosita in regasirea datelor in baze de date este *SELECT* (poate contine sub-clauze).

De obicei, in vorbirea curenta, nu se foloseste denumirea de sub-clauza, ci tot cea de clauza.

Clauza SELECT

- Sintaxa completa a unei clauze *SELECT* diferita de la un sistem la altul

```
SELECT <lista_campuri>
[INTO <numele_noului_tabel>]
FROM <lista_tabele>
[WHERE <conditii> ]
[GROUP BY <lista_campuri> ]
[HAVING <conditii> ]
[ORDER BY <lista_ordonare> [ASC | DESC] ]
```

- In care:
  - lista_campuri - contine coloanele ce vor aparea in setul de rezultate
  - *INTO* <numele_noului_tabel> - indica faptul ca setul de rezultate este utilizat intr-un tabel nou care se si creeaza cu aceasta ocazie
  - lista_tabele - reprezinta tabelele din care provin coloanele ce apar in lista_cumparaturi

  Observatii
- Trebuie respectata ordinea de utilizare a clauzelor
- Nu toate aceste clauze se folosesc tot timpul impreuna si nici toate odata

Sub-clauza (clauza) FROM

- Este obligatorie in orice instructiune *SELECT*
- Poate specifica:
  - Unul sau mai multe tabele
  - Una sau mai multe vederi
  - Asocieri intre doua tabele sau vederi
  - Unul sau mai multe tabele derivate ce contin clauza *SELECT* in interiorul clauzei *FROM*

Sub-clauza (clauza) WHERE

- Folosind constructia

```
SELECT * FROM Note;
```

- Se returneaza toate liniile dintr-un tabel
- Folosind cuvantul cheie *WHERE*, se poate specifica o conditie care sa reduca numarul de linii returnate in setul de inregistrari
- In clauza *WHERE* se pot folosi operatori de comparare sau predicate pentru a verifica o conditie
- Uneori, aceasta clauza mai este folosita pentru a reprezenta o jonctiune dintre doua tabele (corectm dar nerecomandabil)
  Exemplu

```
SELECT * FROM Note
    WHERE Nota > 8;
```

Operatori de comparare

- $>$ (mai mare decât)
- $<$ (mai mic decât)
- $=$ (egal)
- $<=$ (mai mic decât sau egal)
- $>=$ (mai mare decât sau egal)
- $!=$ sau $<>$ (diferit de)
- $!<$ (nu mai mic decât)
- $!>$ (nu mai mare decât)

Sub-clauza (clauza) ORDER BY

- Clauza *ORDER BY* pune la dispozitie o metoda de ordonare a rezultatelor operatiilor
- Daca nu a fos definita o cheie primara rezultatele vor aparea in ordinea introducerii liniilor
- In mod implicit rezultatele unui set sunt ordonate crescator
- Subinterogarile si definitiile vederilor nu pot include clauza *ORDER BY*
- Uneori ar putea fi utila folosirea clauzei *ORDER BY* pentru mai multe campuri
- Clauza *ORDER BY* poate avea coloane care sa nu apara in lista de selectie
- Daca clauza *SELECT* este utilizata impreuna cu clauzele *DISTINCT* sau *UNION*, atunci coloanele dupa care se face ordonarea trebuie sa apara si in lista de selectie
  Exemplu

```
SELECT * FROM Note
    ORDER BY Nota DESC;
```

Sub-clauza (clauza) GROUP BY

- Formeaza grupuri de inregistrari ale unei relatii, pe baza valorilor comune ale unui atribut
- In clauza *GROUP BY* trebuie specificate numele coloanelor din tabelul sau vedere careia le apartin
- De multe ori, clauza *GROUP BY* este utilizata atunci cand se folosesc functii agregat in clauza *SELECT*
- **Componente**

> - Una sau mai multe expresii care se refera la coloane dupa care se poate face o grupare de rezultate expuse in setul afisat
> - Cuvantul cheie, optional, *ALL* care specifica faptul ca in rezultatul final vor aparea toate grupurile produse de clauza *GROUP BY*, chiar daca unele grupuri nu indeplinesc conditia impusa
> - Daca in interogarea de mai sus nu ar aparea clauza *GROUP BY* atunci MS SQL Server ar genera un mesaj de eroare
> - Grupurile de rezultate pot fi afisate cu ajutorul une expresii atata timp cat aceasta nu contine functii agregat
> - In clauza *GROUP BY* trebuie specificat numele coloanei din tabelul sau vederea respectiva si nu denumirea inlocuitoare ce se introduce prin *AS*
> - Daca in loc de *GROUP BY DATEPART (yy, Data_inceput)* s-ar scrie *GROUP BY An* s-ar genera un mesaj de eroare
> - Un tabel poate fi grupat dupa orice combinatie de coloane (in clauza *GROUP BY* se pot utiliza mai multe nume de coloane dintr-un tabel)
> - In clauza *SELECT* nu se folosesc coloane care au mai multe valori pentru aceeasi valoare a coloanei utilizate in clauza *GROUP BY*
> - Afirmatia inversa nu este avevarata

Formatul general

```
SELECT <lista_campuri>
    FROM <nume_tabel>
    GROUP BY <lista_campuri_grupare>
```

Exemple

```
SELECT CodS, AVG(Nota)
    FROM Note
    GROUP BY CodS, An;
```

```
SELECT DATEPART (yy, Data_inceput) AS An, COUNT (*)
AS Numar_studenti
    FROM Student
    GROUP BY DATEPART (yy, Data_inceput);
```

Sub-clauza (clauza) HAVING

- Este asemanatoare utilizarii clauzei *WHERE*
- Permite specificarea unor conditii de selectie care se aplica la nivel de seturi de inregistrari obtinute prin aplicarea clauzei *GROUP BY*
- Din rezultat sunt eliminate toate grupurile care nu satisfac conditia specificata

Formatul general:

```
SELECT <lista_campuri>
    FROM <nume_tabel>
    GROUP BY <lista_coloane_grupare>
    HAVING <conditie_grup>
```

Exemple:

- Se foloseste tabelul Note pentru gruparea studentilor pe ani de studii si afisarea mediei pentru fiecare an in parte

```
SELECT CodS, AVG (Nota)
    FROM Note
    GROUP BY CodS, An;
```

- Daca se modifica aceasta interogaren pentru a returna numai acei studenti care au media peste o anumita valoare

```
SELECT CodS, AVG (Nota)
    FROM Note
    WHERE AVG(Nota)>9
    GROUP BY CodS, An;
```

- Se va obtine un mesaj de eroare(*WHERE* nu opereaza cu functii totalizatoare)
- O solutie corecta ar fi:

```
SELECT CodS, AVG (Nota)
    FROM Note GROUP BY Cods,An
```

```
    HAVING AVG(Nota)>9
```

- Clauza *HAVING* permite utilizarea functiilor totalizatoare intr-o instructiune de comparare, asigurand pentru functiile totalizatoare ceea ce *WHERE* asigura pentru linii individuale
- Rezultatul interogarii urmatoare:

```
SELECT CodS, AVG (Nota)
    FROM Note GROUP BY Cods,An,Nota
    HAVING Nota >9
```

- Este diferit de precedentul
- Constructia "*HAVING AVG (Nota) >9*" a evaluat fiecare grup in parte si a transmis la iesire numai grupurile care aveau o medie mai mare de 9, adica exact ceea ce se dorea
- Constructia "*HAVING (Nota) >9*" conduce la un rezultat diferit
- Daca utilizatorul cere evaluarea si afisarea grupurilor de medii care respecta conditia "*Nota >9*", ar trebui examinat fiecare grup in parte si eliminate acelea in care exista o nota mai mica de 9
- Unele versiuni de SQL returneaza un mesaj de eroare in cazul in care se foloseste altceva decat o functie totalizatoare intr-o clauza *HAVING*
- In cacuza *HAVING* se pot folosi si mai multe conditii (separate prin *AND*, *OR* sau *NOT*)

```
SELECT COUNT(CodS), AVG(Nota)
    FROM Note
    GROUP BY An HAVING COUNT(CodS)>1 AND AVG(Nota)>5
```

- In clauza *HAVING* se poate folosi o functie totalizatoare care nu a fost folosita in instructiunea *SELECT*
- Care sunt studentii care au primit note in anul 1

```
SELECT CodS
    FROM Note
    GROUP BY Cods,An
    HAVING COUNT(*) > 0 AND An='1'
```

- Operatorul *IN* opereaza corect intr-o clauza *HAVING*
- In clauza *HAVING* se pot utiliza si subinterogari

Subinterogari

- Subinterogarile orefa un mijloc de a baza o interogare pe o alta (imbricarea interogarilor)

Reguli

Lista de selectie a unei subinterogari create cu ajutorul unui operator de comparare poate contine doar o singura expresie sau nume de coloana

Daca clauza *WHERE* a unei interogari exterioare contine numele unei coloane, aceasta trebuie sa fie compatibila din punct de vedere al jonctiunii cu coloana din subinterogare

Tipurile de date *ntext*, *text* si *image* nu sunt permise intr-o subinterogare

Clauza *DISTINCT* nu poate fi utilizata in subinterogari ce contin clauza *GROUP BY*

Intr-o subinterogare nu poate fi utilizata clauza *ORDER BY* decat daca s-a utilizat si restrictia *TOP*

O vedere creata pe baza unei subinterogari nu poate fi actualizata

Observatii

Constructiile *SELECT* pot fi imbricate pe max. 16 nivele

O subinterogare difera de operatorul de cuplare prin aceea ca rezultatul final contine date doar de la ultimul tabel

Subinterogare

O clauza *SELECT* care este inserata in cadrul altei cauze *SELECT*

Se foloseste, de obicei, in cadrul unei clauze *WHERE*

Varianta 1

Predicatele [ANY | SOME | ALL]
Exemplu
Urmatoarea constructie intoarce toti studentii (si notele corespunzatoare acestora) care au cele mai mari note

```
SELECT Nume, Prenume, Nota
    FROM Student INNER JOIN Note ON
    Student.CodS=Note.CodS
    WHERE Nota = (SELECT Max (Nota) FROM Note);
```

Ceea ce conduce la un rezultat diferit de varianta:

```
SELECT Nume, Prenume, MAX(Nota)
    FROM Student INNER JOIN Note ON
    Student.CodS=Note.CodS
    GROUP BY Nume, Prenume;
```

In care raspunsul corespunde intrebarii: "Care este cea mai mare nota obtinuta de fiecare student?"

Se observa faptul ca, deoarece subinterogarea intoarce o singura valoare, nu este nevoie sa se foloseasca nici unul dntre predicatele *ANY*,*SOME* sau *ALL*

Urmatoarea constructie prezinta toti studentii (si notele corespunzatoare acestora) care au note mai mari decat notele studentului Radu Tiberiu:

```
SELECT Nume, Prenume, Nota
    FROM Student INNER JOIN Note ON Student.Cods=Note.Cods
    WHERE Nota > All(
    SELECT Nota
        FROM Note INNER JOIN Student ON Student.Cods =Note.Cods
        WHERE Nume = 'Radu' And Prenume='Tiberiu');
```

Observatii

- Se observa faptul ca predicatele *ANY* si *SOME* conduc la obtinerea aceluiasi rezultat, returnand toate variantele ce respecta conditita de comaprare pentru cel putin una dintre valorile returnate de catre subinterogare
- Daca se inlocuieste *ALL* cu *SOME* in interogarea anterioara, rezultatul obtinut va contine toti studentii (si notele corespunzatoarea acestora) care sunt mai mari decat cea mai mica nota acordata studentului Radu Tiberiu

Varianta 2

Predicatul [NOT] IN

Se foloseste la cautarea unei valori a unei coloane dintr-un tabel rezultat in urma altei interogari

Exemplu

Urmatoarea constructie returneaza toti studentii (si notele corespunzatoare) din tabelul "Student" care nu apar in tabelul "Note" (Studentii care nu au note):

```
SELECT CodS, Nume,Prenume
    FROM Student
    WHERE CodS NOT IN (SELECT CodS FROM Note);
```

Varianta 3

Predicatul [NOT] EXISTS

Se foloseste atunci cand se verifica daca o anumita valoare exista (este returnata) in urma executarii subinterogarii

Exemplu

Urmatoarea constructie afiseaza toti studentii care nu au note:

```
SELECT Cods, Nume, Prenume
    FROM Student
    WHERE NOT EXISTS
        (SELECT * FROM Note WHERE Note.Cods =Student.Cods);
```

Se observa faptul ca tabelul "Note" este referit in subtinterogare, ceea ce face ca SQL Server sa evalueze subinterogarea cate o data pentru fiecarea valoare a indentificatorului "CodS" din tabelul "Note"

Observatii

- Atunci cand se foloseste variante 1 sau 2, subinterogarea trebuie sa returneze o singura coloana, altfel va apare un mesaj de eroare
- Constructia *SELECT* care contine subinterogarea pastreaza acelasi format si reguli ca orice alta constructie *SELECT*, dar trebuie in mod obligatoriu introdusa in paranteze

Sisteme de gestiune a bazelor de date orientate pe obiecte (SGBDOO)

Dezavantajele modelului relational

- Crearea de relatii care nu corespund tipurilor de entitati din lumea reala, datorita procesului de normalizare care impune fragmentarea unei entitati in mai multe relatii, necesitand, in timpul prelucrarii interogarii, formarea de uniuni multiple care sunt operatii costisitoare
- Supraincarcarea semantica a modelului relational, datorita inexistentei unui mecanism care sa poata face distinctia dintre entitati si relatii sau pentru a deosebi diferitele tipuri de relatii care exista intre entitati, ceea ce face sa nu fie posibil sa se construiasca semantica in cadrul operatiilor;
- Operatii limitate doar la cele cu multimi si tupluri, insuficiente pentru a modela comportamentul multor obiecte din lumea reala
- Impunerea de catre modelul relational a unei omogenitati a datelor atat orizontala, prin faptul ca fiecare tuplu al unei relatii trebuie sa fie compus din aceleasi atribute, cat si verticala, prin faptul ca valorile dintr-o coloana a unei relatii trebuie sa provina din acelasi domeniu, dar si a faptului ca la intersectia dintre o coloana si un rand trebuie sa

existe o valoare atomica, ceea ce duce la impunerea prea multor restrictii a uno obiecte complexe din lumea reala;
- Dificultatea in manipularea interogarilor recursive, adica interogarile despre asocierile create de o relatie (direct sau indirect) cu ea insasi;
- Dificultati referitoare la concurenta, modificarile schemei, accesul navigational
- Nu se permite descrierea structurilor de date complexe (date in format multimedia, documente electronice, grafice etc.);
- Nu se permite definirea unor tipuri de date definite de utilizator;
- Nu se permite partajarea/reutilizarea structurilor de date;
- Nu se permite declararea prelucrarilor aferente structurilor de date deoarece datele sunt decise separat de prelucrari
- **Sistemele de gestiune a bazelor de date relationale nu pot fi aplicate in cazul aplicatiilor, cum ar fi:**
  - Proiectarea asistata de calculator
  - Fabricarea asistata de calculator
  - Ingineria programarii asistata de calculator
  - Sisteme multimedia
  - Editare digitala
  - Sisteme informationale geografice
- **Modelarea orientata pe obiect se bazeaza pe urmatoarele concepte**
  - Obiect
  - Abstractizare
  - Incapsulare
  - Mostenire
  - Poliformism

Obiectul

- Este o entitate cu identitate proprie caracterizat prin stare si comportament

Identitatea

- Este proprietatea acestuia care l distinge de alte obiecte si este de regula o adresa logica invatianta (pointer)
- Identificarea obiectelor este asigurata automat de sistem si este transparenta utilizatorului
- Flecare obiect are un identificator unic pentru intregul sistem, care este independent de valorile atributelor sale si, ideal, este vizibil pentru utilizatori
- Doua obiecte O_1 si O_2 sunt identice (se noteaza $O_1 == O_2$) daca au acelasi identificator, adica egalitatea obiectelor este de fapt o egalitate de pointeri
- Doua obiecte O_1 si O_2 sunt egale (se noteaza $O_1 = O_2$) daca au aceleasi valori
- Se poate spune ca identitatea obiectelor implica egalitatea lor, reciproca fiind falsa

Starea

- Este definita de valorile atributelor sale
- Un atribut este definit printr-un nume si poate lua valori elementare (numeric, alfanumeric etc.) sau complexe (structuri de valori multiple, referinte spre alte obiecte, tipuri utilizator etc.)
- Atributele se pot concretiza in locatii de memorie in care sunt stocate datele aferente

Exemplu

```
Studentul S1: CodS = “001”
              Nume =”Banu”
              Prenume = “Andrei”
```

Comportamentul

- Este definit de setul de operatii (metode) aplicabile obiectului respectiv
- Metodele pot fi utilizate pentru a modifica starea obiectului, prin modificarea valorilor atributelor, sau pentru interogarea valorilor atributelor selectate
- Operatiile unui obiect constituie de regula, modalitati de raspuns la mesajele primite din exterior
- Mesajele sunt mijloace prin care comunica obiectele
- Un mesaj reprezinta o cerere a unui obiect (emitent) catre alt obiect (receptor), prin care cel de-al doile obiect este solicitat sa execute una dintre metodele sale
- Emitentul si receptorul pot reprezenta acelasi obiect
- O operatie este definita printr-o *semnatura* compusa din denumire, un set optional de parametrii de apel si un set optional de parametrii de retur
- Operatiile sunt de regula proceduri sau functii ce actioneaza asupra atributelor obiectului respectiv

Obiectul

Abstractizarea

- Este procesul prin care obiectele cu aceleasi atribute si cu un comportament comun sunt grupate in tipuri abstracte de date numite *clase*
- Obiectele devin realizari (instantieri) ale claselor
- Clasele sunt reprezentari (modele) abstracte, conceptuale ale entitatilor descrise de obiecte

Incapsularea

- Consta in capacitatea obiectelor (claselor) de a contine la un loc atat date cat si metode dintre care numai o parte sunt vizibile din exterior
- Obiectele/Clasele apar ca niste cutii negre care ascund detaliile de implementare, oferind in schimb o interfata pentru acces

Exemplu

- Se considera clasa *Student* descrisa prin atributele invizibile (private) din interiorul clasei

```
CodS, Nume, Prenume, Data_inceput, Data_sfarsit, Sex
```

- Si operatiile accesibile din exterior (publice)

```
Creeaza(CodS, Nume, Prenume, DataInceput, DataSfarsit, Sex)
ReturneazăCodS(): text
ReturneazăNume(): text
ReturneazăPrenume(): text
ReturneazăDataInceput(): data
ReturneazăDataSfarsit(): data
ReturneazăSex(): text
```

- Datele studentului "Banu Andrei" sunt atribuite numai la crearea instantei (obiectului) respectiv, prin apelarea metodei-constructor vizibila din interiorul obiectului

```
Creeaza(“001”, “Banu”, “Andrei”, “01-11-2002”,“b”)
```

- Dupa ce a fost creat un obiect, datele aferente studentului respectiv nu mai pt fi modificare, ci numai citite prin intermediul metodelor de tip "Returneaza"
- Utilizatorul va percepe un obiect *Student* numai prin datele *read-only*
- Este exclusa posibilitatea modificarii unei persoane ulterior crearii sale
- Se asigura astfel si o securitate a datelor stocate de un obiect impotriva modificarii neautorizate sau/si accidentale

Comparatie intre modelul obiectual si cel relational

Modelul O.O.	**Modelul Relational**
Clasa	Tabel(relatie)
Obiect	Tuplu(inregistrare)
Atribut	Camp
Operatie(metoda)	-

Mostenirea

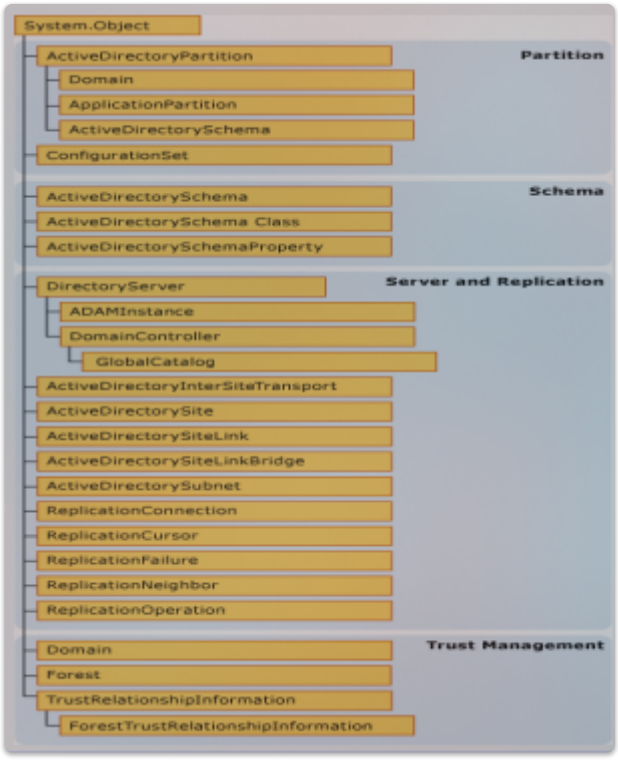
- Este procesul prin care toate atributele si metodele publice (vizibile din exterior) ale unei clase, numita clasa de baza, sunt preluate automat de o alta clasa inrudita, numita clasa derivata sau subclasa
- Clasele derivate pot contine si atribute sau metode specifice

- Permite ca o clasa sa fie definita ca un caz special, numit subclasa a unei clase mult mai generale, numita superclasa
- Este o tehnica prin care se incurajeaza reutilizarea/partajarea datelor

Relatii intre clase

- Clasificare
- Generalizare - procesul de formare a unei superclase
- Specializare - procesul de formare a uneu subclase

Clasificare

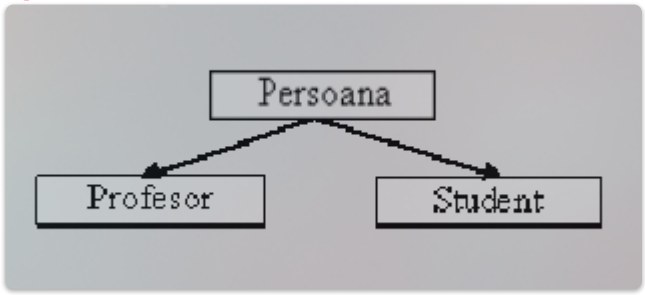
- **Exemplu:**
  - NET Framework
  - 

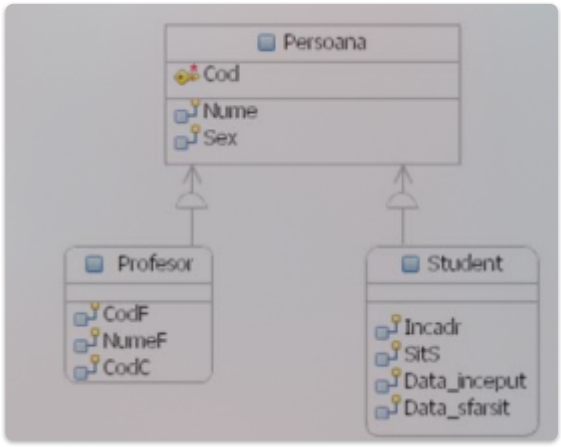
Generalizare - specializare

- O clasa de la care se mosteneste se numeste clasa de baza
- Clasele care mostenesc se numesc clase derivate
- O clasa derivata are toate variabilele, metodele, proprietatile, operatorii preluati de la clasa de baza
- O clasa derivata adauga elemente proprii
- Membrii cu caracter privat nu se mostenesc
- **Exemplu**

```
class <nume clasa derivata>: <nume clasa de baza>
{
    //corp clasa
}
```

Principiul de substituire

- O instanta a subclasei poate fi utilizata ori de cate ori o metoda sau o constructie asteapta o instanta a superclasei corespunzatoare
- **Exemplu**
  - 

  - 

  - Se considera clasa "Persoana" definită prin atributele publice (vizibile din afara clasei): “CNP”, “Nume”, “Prenume”, “Data_nasterii”, “Sex” şi metoda publică “Retrunează Varsta(AnReferinta)”
  - Clasa “Persoana” poate fi derivată în clasele “Profesor” şi “Student”, de acelaşi tip cu prima, astfel încât fiecare student sau profesor va fi descris prin CNP, nume, prenume, sex şi vârsta calculată în funcţie de un anumit an de naştere
  - Clasa “Student” poate conţine ca atribute specifice “Curs”, “Nota”, “An”, “Taxa” precum şi metoda proprie “Media()”
  - Clasa “Profesor” poate fi descrisă prin atributele proprii “CodP”, “Data_angajarii” şi metoda “Returnează Vechime()”

Caracteristici

- Este procesul prin care un obiect poate sa preia prototipul altui obiect
- Este o tehnica de programare prin care se pot reutiliza si extinde clasele existentem avand avantajul de a nu fi necesara rescrierea codului original
- Constituie diferenta esentiala dintre limbajele conventionale si clasele din limbajele orientate pe obiecte
- O clasa derivata mosteneste toate caracteristicile (starea si comportamentul) clasei de baza
- Scopul claselor derivate este acela de a adauga noi caracteristici unei clase de baza

- Dacă codul, o procedură sau o funcţie sunt definite pentru clasa de obiecte Persoana, iar clasa Profesor moşteneşte de la clasa Persoana, atunci codul este definit şi pentru subclasa Profesor (cu excepţia redefinirii procedurilor sau funcţiilor în subclasa Profesor).
- Moştenirea suportă refolosirea codului în următorul mod pentru acest exemplu: dacă se creează clasa Persoana şi aceasta este suficient de apropiată pentru a satisface o anumită cerinţă, se încorporează Persoana, se introduce Profesor ca subclasă a clasei Persoana şi se fac adăugiri la clasa Profesor pentru a îndeplini funcţionalitatea cerută
- Se spune că Profesor este o subclasă a clasei Persoana atunci când fiecare proprietate aplicabilă oricărei instanţe obiect a clasei Persoana este aplicabilă oricărei instanţe obiect a clasei Profesor
- Suplimentar, se acceptă adiţionarea la Profesor astfel încât să se poată rescrie funcţionalităţi disponibile în clasa Persoana
- Dacă clasa Profesor face numai adiţionări semantice la clasa Persoana se spune că Profesor este o extensie conservativă a clasei Persoana
- Specializarea reprezintă diferenţele de stare sau comportament faţă de clasa de bază şi poate însemna:
  - Ignorarea unor caracteristici ale clasei de bază (rar întâlnită);
  - Adăugarea de noi caracteristici (variabile de stare sau metode), numită specializare prin îmbogăţire;
  - Modificarea unor caracteristici ale clasei de bază (de obicei se modifică metode), numită specializare prin înlocuire
- Moştenirea permite ca în clasa derivată să se specifice doar caracteristicile noi
- Definitia clasei derivate contine
  - Precizarea parintelui (parintilor)
  - Precizarea noilor caracteristici
- Acest lucru ofera avantajele
  - Rezultarea definitiilor si a codului caracteristicile mostenite de la parinti nu mai trebuie nici specificate, nici codificate;
  - Definiti mai simple ale claselor derivate prin specificarea a mai putine elemente; se inteleg mai bine specificarea si implementarea

Tipuri de mostenire

- Mostenirea simpla, corespunzatoare unei singure clase
- Mostenire multipla, care presupune existenta a cel putin doi parinti

Relatia de mostenire

- Este o legatura intre clase, de la parinte la fiu, ce stabileste o relatie (partiala) de ordine pe multimea claselor
- In cazul mostenirii simple ierarhia claselor se poate reprezenta sub forma unui arbore (arbore de mostenire, un nod corespunzand unei clase)
- Arborele are ca radacina clasa de baza a ierarhiei, cea mai generala clasa, care prezinta caracteristicile comune ale tuturor claselor din ierarhie

- In cazul mostenirii multiple ierarhia claselor se reprezinta sub forma unui graf, graful de mostenire
- In arborele de mostenire pot exista subarbori, in care radacina este clasa de baza a subarborelui
- Relatia poate fi de tipul
  - Unul-la-multi, in cazul mostenirii simple (unui parine ii corespund mai multi descendenti);
  - Multi-la-multi, in cazul mostenirii multiple (unui parine ii corespund mai multi descendenti, iar un descendent poate avea mai multi parinti)
- In ierarhia de mostenire, o clasa aflata intr-un mod intern sau terminal are doua tipuri de caracteristici
  - Caracteristici mostenite de la parinti
  - Caracteristici proprii, specificate in definitia ei (fie caracteristici noi, fie redefiniri ale caracteristicilor parintilor)
- Schema de specificare a unei clase trebuie sa contina, pe langa elementele preluate (nume, atribute, metode) si precizarea parintilor (superclaselor) clasei respective, sub forma:

```
Clasa
    Nume
Superclasa
    Lista de superclase
Atribute
    Specificarea atributelor
Metode
    Specificarea metodelor
```

- **Exemplu**
- Fie C o clasa si A un atribut al ei
- Daca A este mostenit, se pune problema determinarii superclasei SC a lui C pentru care A este caracteristica proprie (SC va fi clasa de la care C mostenteste caracteristica A)
- In cazul unei mosteniri simple, exista un singur drum de la radacina arborelui de mostenire la nodul corespunzator clasei C, relatia de mostenire inducand o relatie de ordine pe multimea claselor existente in nodurile acestui drum
- Identificarea superclasei SC se face prin parcurgerea respectivului drum in sens invers si inspectarea fiecarui nod intalnit
- Primul nod in care se gaseste definitia atributului A va corespunde clasei SC
- In cazul mostenirii multiple, exista mai multe drumuri de la nodul clasei C la nodul initial al grafului de mostenire
- Pot exista doua situatii
  - Atributul A aparţine unui singur drum (C moşteneşte atributul A de la un singur părinte); determinarea clasei SC se face pe acest drum, la fel ca în cazul moştenirii simple;

    - Atributul A aparţine la cel puţin două drumuri (C moşteneşte atributul A de la cel puţin doi părinţi), situaţie numită conflict de moştenire; în acest caz trebuie (pe baza unei informaţii suplimentare) precizat părintele de la care C moşteneşte atributul A
- Pentru rezolvarea conflictului se folosesc:
    - Stabilirea unei ierarhi între părinţi (această ierarhie va dicta ordinea în care se iau în considerare drumurile);
    - Conflictele sunt de fapt conflicte de nume; se poate încerca o schimbare a numelor sau o calificare a lor cu numele clasei părinte (neelegant);
    - Moştenirea se specifică explicit, în genul: "moşteneşte A de la SC"

Relatia de instantiere

- Este o legatura intre obiecte si clase: obiectul este o instanta a unei singure clase (legatura de tipul unu-la-unu)
- O clasa poate avea mai multe instante
- Intr-o ierarhie de clase, nu toate clasele au instante

Categorii de clase

- **Clase abstracte**, ce nu genereaza instante; de obicei, ele sunt in partea superioara a ierarhiei (clase de baza), continand caracteristicile comune ale tuturor claselor descendente;
- **Clase generoatoare de instante**, ce se afla in nodurile interoare sau terminale ale ierarhiei

Clase abstracte

- Sunt destinate, de obicei, definirii metodelor mestenite de catre subclase
- Clasele care mostenesc metodele le pot rafina si dezvolta

Proprietatile claselor abstracte

- Genericitatea, care inseamna acelasi comportament pentru clase diferite;
- Parametrizarea, care permite utilizarea unei clase generice in cazuri particulare (instantarea unei clase generice)

Tehnici speciale de divizre a metodelor

- Subrutinele, care constituie calea cea mai simpla, codul comun fiind extras intr-o metoda care este apelata de fiecare dintre metodele initiale. Metoda comuna poate fi atribuita unei superclase comune;
- Factorizarea, adica partea comuna constituie o noua metoda care apeleaza o operatie-implementata prin metode diferite, ce contin diferentele de cod si sunt alocate unor subclase. Uneori, este posibila adaugarea unei clase abstracte care sa contina metoda de nivel inalt

Polimorfism

- Inseamna mai multe forme, aspecte, infatisari
- Abilitatea de a
  - Pune obiecte inrudite intr-un tablou sau colectie
  - Utiliza protocoul de comunicatie pentru a transmite mesaje obiectelor individuale printr-o referire unitara (ca elementele de tablou sau colectie)
- Permite unei interfete sa fie folosita cu o clasa generala de actiuni
- Actiunea specifica selectata (adica metoda) este determinata de natura precisa a situatiei, iar selectia se face cu ajutorul compilatorului care este responsabil de acesta
- Intr-un program pot exista obiecte diferite, care sa fie instante ale unor clase legate intre ele prin relatia de mostenire (obiecte inrudite/polimorfice)
- Obiectele polimorfice reprezinta instante obiect care pot avea diferite forme in timpul rularii programelor (in cazul acesta nu se stabileste legatura dintre obiect si metode sale in faza de compilare ci numai in faza de executie)
- Obiectele polimorfice permit procesarea obiectelor al caror tip nu este cunoscut in momentul compilarii
- Polimorfismul poate fi asigurat prin doua cai:
  - Redefinirea (rescrierea) metodelor mostenite in clasele derivate;
  - Supraincarcarea unie metode in cadrul aceleiasi clase (crearea unor metode cu acelasi nume, dar cu parametrii diferiti)

  Exemplu
- Fie clasa "Student" definita prin atributele

```
"CodS", “Nume”, “Prenume”, “Data_inceput”, “Data_sfarsit”, “Situatie”, “Inv”
```

- Si metodele asociate

```
“Sterge(Cods: text)”
“Sterge(Nume:text, Prenume:text)
```

- Clasa se va comporta diferit la primirea mesajului de stergere (polimorfism prin supraincarcarea metodei "Sterge")
  - Daca mesajul contine un parametru de tip intreg (codul studentului), atunci se va elimina obiectul student ce va avea acest cod
  - Daca cererea de stergere contine doi parametri de tip sir de caractere, atunci va fi eliminat studentul cu numele respectiv

Caracteristicile obiectelor polimorfice

- Ierarhi claselor ale caror instante sunt o clasa radacina, clasa de baza

- Clasa de baza defineste protocolul de comunicatie comun tuturor obiectelor inrudite (toate obiectele sunt capabile sa raspunda la aceleasi mesaje)
- Sunt de tipuri diferite (instante de clase diferite)
- Polimorfismul necesita mostenire; fara mostenire nu ar exista clasa de baza si, deci, nici protocol de comunicatie;
- Mostenirea nu este suficienta pentru realizarea polimorfismului, fiind nevoie de mecanisme suplimentare;
- Polimorfismul semnifica posibilitatea unuo obiect, instanta a unei clase, sa raspunda diferit la primirea unui mesaj
- Polimorfismul mareste flexibilitatea modelului orientat pe obiect in reprezentarea cat mai sintetica a realitatii

Diferentele dintre polimorfism si mostenire

- Mostenirea are ca scop ierarhizarea claselor (tipurilor de date) in ideea unei mai bune structurari a obiectelor, prin eliminarea redundantelor, in timp ce polimorfismul simplifica comunicarea cu sau intre obiectele inrudite
- Mostenirea implica toate caracteristicile claselor (atribute si metode), pe cand polimorfismul are ca obiect doar metodele ce definesc protocolul de comunicatie (numai metodele virtuale)
- Mostenirea are ca efect reutilizarea codului si permite manifestarea polimorfismului; polimorfismul utilizeaza mostenirea pentru a constru ierarhii de tipul polimorfice, ce au in comun acelasi protocol de comunicatie, definit in clasa de baza

Sisteme de gestiune a bazelor de date relationale orientate pe obiecte (SGBDROO)

- Sunt extensii ale modelului relational
- Principala imunatatire adusa modelului relational o constituie tipurile abstracte de date (TAD) care permit
  - Partajarea datelor
  - Crearea de tipuri de date definite de utilizator;
  - Utilizarea structurilor de date complexe;
  - Incapsularea
  - Mostenirea

Produse reprezentative

- ORACLE
- DB2

Tipuri abstracte de date - TAD

- Tipuri de date definite de utilizator, diferite de tipurile implicite numeric, sir de caractere, etc. ce pot fi utilizate la definirea tabelelor

- Clasa admite instaniteri directe, in timp ce un TAD serevste numai ca model la definirea structurii unuia sau mai multor tabele (partajarea TAD-ului)
  Exemplu
- Fie TAD-ul

```
Persoana(CNP,Nume,Prenume,Sex)
```

- Acesta poate servi drept structura pentru tabelele "Profesor" si "Student"

```
Profesor(Pers: Persoana, Salariu: real)

Student(Pers:Persoana, Data_inceput:date, Data_sfarsit:date,Situatie:text, Ir
```

- Reprezinta faptul ca un camp din structura sa poate fi un tip elementar (numeric, sir de caractere etc.), un alt TAD sau o colectie
- Imbricarea TAD-urilor se poate realiza fie direct, fie prin intermediul pointerilor
- Fie TAD-ul

```
Persoana(CNP:text, Nume:text, Prenume:text, Adresa:text)
```

- Acest tip poate fi utilizat la definirea TAD-ului Profesor in doua variante

```
1. Profesor(Pers:Persoana, Salariu:real) - în care tipul Profesor conţine ele
2. Profesor(Pers:REF Persoana, Salariu:text) - în care tipul Profesor conţine
```

- Pentru a reprezenta copiii unui profesor se poate recurge la folosirea unei colectii (set) de tip Persoana:

```
Profesor(Pers:REF Persoana, Salariu:text, Copii SET(Persoana))
```

Caracteristici TAD

Incapsulare

- TAD-urile pot sa contina atat date/campuri cat si metode/operatii implementate prin proceduri sau functii utilizator

Mostenire

- Consta in posibilitatea derivarii unui TAD generic in TAD-uri specializate
- Prin mostenire, un TAD derivat va prelua structura (datele) si metodele TAD-ului de baza
- Un TAD derivat poate fi descris si prin date si metode specifice

  Exemplu
- Fie TAD-ul

```
Persoana(CNP, Nume, Prenume, Adresa)
```

- Acest tip generic poate fi derivat in doua subtipuri

```
Profesor Under Persoana(Vechime, Salariu)
Student Under Persoana(An, Situatie, Inv)
```

Observatii

-Schemai unui SGBDROO contine schemele TAD-urilor, ale tabelelor, relatiile dintre acestea precum si restrictiile de integrare