CUNOȘTINȚE DE SPECIALITATE

INFORMATICĂ ECONOMICĂ

CAP.1. BAZE DE DATE – Conf.univ.dr. ANCA MEHEDINȚU

CAP.2. PROGRAMARE ORIENTATĂ OBIECT ŞI STRUCTURI DE DATE – Prof. univ.dr. GEORGETA ȘOAVĂ și Conf.univ.dr. AMELIA BĂDICĂ

CAP.3. SISTEME DE OPERARE – Conf.univ.dr. SORIN POPA

CAP.4. PROIECTAREA SISTEMELOR INFORMATICE – Conf.univ.dr. RĂZVAN BUȘE

CAP.5. PROGRAMARE ÎN INTERNET- Conf.univ.dr. AMELIA BĂDICĂ

2015

CUPRINS

CAP.1. BAZE DE DATE

1.1. CONCEPTE ALE BAZELOR DE DATE RELAŢIONALE

1.1.1. Definiţii

Baze de date

O bază de date este o colecţie de informaţii interrelaţionate gestionate ca o singură unitate. Această definiţie este intenţionat foarte largă, deoarece există mari diferenţe între concepţiile diferiţilor producători care pun la dispoziţie sisteme de baze de date. De exemplu, Oracle Corporation defineşte o bază de date ca fiind o colecţie de fişiere fizice gestionate de o singură instanţă (copie) a produsului software pentru baze de date, în timp ce Microsoft defineşte o bază de date SQL Server ca fiind o colecţie de date şi alte obiecte. Un *obiect* al bazei de date este o structură de date denumită stocată în baza de date, cum ar fi un tabel, o vizualizare sau un index.

Sisteme de gestiune a bazelor de date

Un *sistem de gestionare a bazei de date (DBMS - database management system)* este un produs software furnizat de producătorul bazei de date. Produse software precum Microsoft Access, Microsoft SQL Server, Oracle Database, Sybase, DB2, INGRES, MySQL şi PostgreSQL fac parte din categoria DBMS sau, mai corect, *DBMS relaţionale (RDBMS).*

Sistemul DBMS pune la dispoziţie toate serviciile de bază necesare pentru organizarea şi întreţinerea bazei de date, inclusiv:

- Transferarea datelor în şi din fişierele fizice de date, în funcţie de cerinţe.
- Gestionarea accesului concurenţial la date al mai multor utilizatori, inclusiv prevenirea conflictelor care ar putea fi cauzate de actualizările simultane.
- Gestionarea tranzacţiilor, astfel încât toate modificările făcute asupra bazei de date printr-o tranzacţie să fie executate ca o singură unitate. Cu alte cuvinte, dacă tranzacţia reuşeşte, toate modificările efectuate de tranzacţie sunt înregistrate în baza de date; dacă tranzacţia eşuează, nici una dintre modificări nu este înregistrată în baza de date. Totuşi, unele sisteme RDBMS nu asigură suportul pentru tranzacţii.
- Acceptă un *limbaj de interogare,* care reprezintă sistemul de comenzi folosit de utilizator pentru a obţine date din baza de date. SQL este principalul limbaj folosit pentru sistemele DBMS relaţionale.
- Funcţii pentru salvarea bazei de date şi pentru refacerea bazei de date în urma erorilor.
- Mecanisme de securitate pentru împiedicarea accesului neautorizat la date şi modificarea acestora.

Baze de date relaţionale

O *bază de date relaţională* este o bază de date care respectă modelul relaţional, dezvoltat de Dr. E. F. Codd. Modelul relaţional prezintă datele sub forma familiarelor tabele bidimensionale, similar cu o foaie de calcul tabelar Excel. Spre deosebire de o foaie de calcul tabelar, nu este obligatoriu ca datele să fie stocate într-o formă tabelară, iar modelul permite şi combinarea tabelelor *(crearea uniunilor (joining),* în terminologia relaţională) pentru formarea vizualizărilor, care sunt prezentate tot ca tabele bidimensionale. Flexibilitatea extraordinară a bazelor de date relaţionale este dată de posibilitatea de a folosi tabelele independent sau în combinaţii, fără nici o ierarhie sau secvenţă predefinită în care trebuie să se facă accesul la date.

1.1.2. Componente ale bazelor de date relaţionale

1. Tabele

Unitatea primară de stocare a datelor într-o bază de date relaţională este ***tabelul,*** care este o

structură bidimensională compusă din rânduri şi coloane. Fiecare tabel reprezintă o *entitate,* ceea ce înseamnă o persoană, un loc, un lucru sau un eveniment care trebuie să fie reprezentat în baza de date, cum ar fi un client, un cont bancar sau o tranzacţie bancară. Fiecare rând al tabelului reprezintă o apariţie a entităţii.

Figura 1.1 reprezintă conţinutul parţial al unui tabel numit **MOVIE** (Film).

Tabelul **MOVIE** este parte a unei baze de date pentru un magazin de produse video, folosită ca exemplu. Tabelul **MOVIE** conţine date care descriu filmele disponibile în magazinul de produse video. Fiecare rând din tabel reprezintă un film, iar fiecare coloană reprezintă o caracteristică a filmului respectiv, cum ar fi titlul filmului.

MOVIE_ID	MOVIE_ GENRE_ CODE	MPAA_ RATING_ CODE	MOVIE_TITLE	RETAIL_PRICE_ VHS	RETAIL_PRICE_ DVD	YEAR_ PRODUCED
1	Drama	R	Mystic River	58,97	19,96	2003
2	ActAd	R	The Last Samurai	15,95	19,96	2003

Fig. 1.1 Structura tabelei Movie (Film)

2. Relaţii

Relaţiile reprezintă asocierile dintre tabelele bazelor de date relaţionale. Deşi fiecare tabel relaţional poate exista independent, esenţa bazelor de date este tocmai stocarea informaţiilor între care există legături. De exemplu, pe lângă filmele propriu-zise, se pot stoca şi informaţii despre categoriile folosite de magazin pentru organizarea inventarelor de filme. În acelaşi timp, puteţi stoca şi informaţii despre copiile fiecărui film, inclusiv data la care a fost primită copia şi formatul acesteia, cum ar DVD sau VHS. Prin folosirea relaţiilor, se pot asocia tabelele înrudite, într-un mod formal, uşor de folosit astfel încât să combinăm date din tabele multiple în aceeaşi interogare a bazei de date, dar păstrând flexibilitatea de a include numai informaţiile care îl interesează pe utilizator. Posibilitatea de a selecta din baza de date numai informaţiile care ne interesează ne permite să ajustăm informaţiile din baza de date în funcţie de cerinţele specifice ale persoanelor sau aplicaţiilor care au acces la baza de date.

Figura 1.2 prezintă patru tabele din baza de date a magazinului de produse video şi relaţiile dintre acestea, într-un format cunoscut sub numele de diagramă de relaţii a entităţilor (ERD - Entity Relationship Diagram).

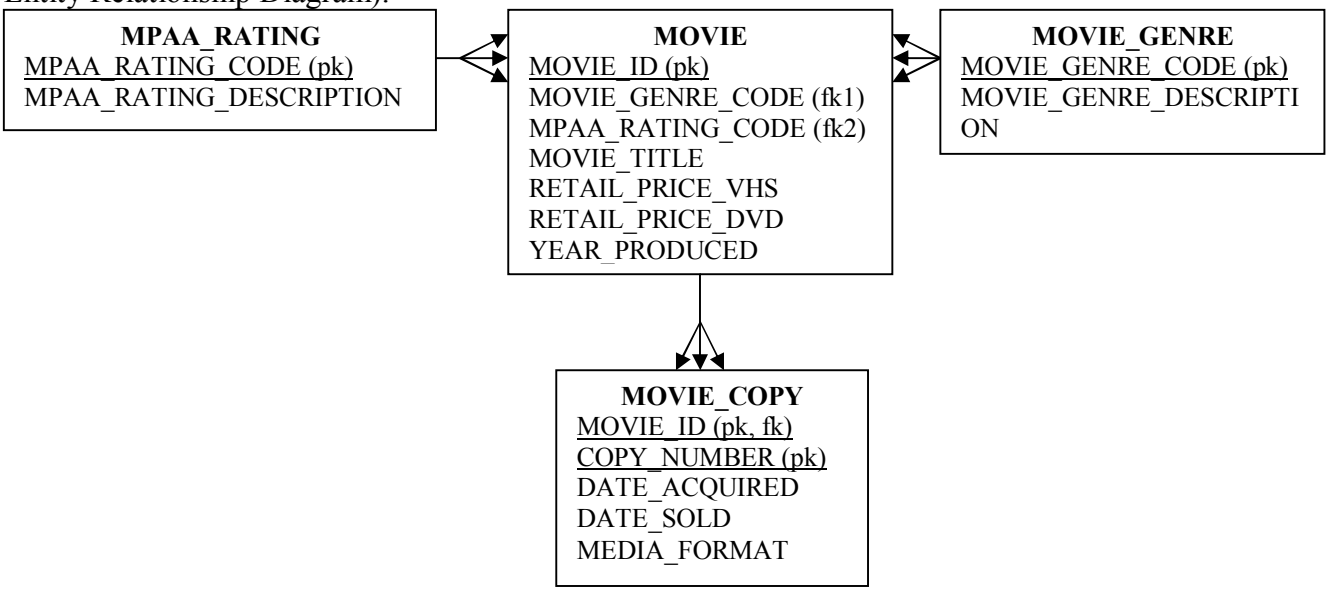
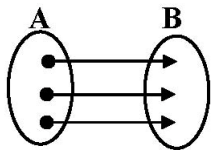


Fig. 1.2 Diagrama ERD a bazei de date pentru magazinul de produse video (prezentare parţială)

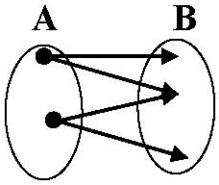
Diagramele ERD ne pun la dispoziţie o modalitate de prezentare a proiectului general al unei baze de date relaţionale, într-un format uşor de înţeles pentru utilizatorii bazei de date, indiferent dacă au sau nu cunoştinţe tehnice. Fiecare dreptunghi din diagramă reprezintă un tabel relaţional, cu numele tabelului scris deasupra liniei orizontale şi coloanele tabelului enumerate pe verticală, în porţiunea principală a dreptunghiului.

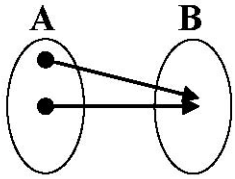
În funcţie de numărul de elemente, între care se stabilesc relaţii, aparţinând celor două colecţii, aceste relaţii pot fi de tip unu la unu, unu la mulţi şi mulţi la mulţi.



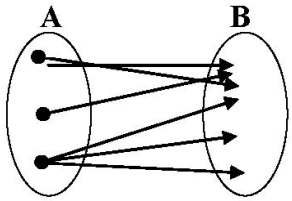
Relaţiile de tipul 1→1 (unu la unu), care presupun că unui membru din colecţia A îi corespunde un singur membru din colecţia B.

Relaţiile de tipul 1→m sau m→1 (unu la mulţi sau mulţi la unu), care presupun că unui membru din prima entitate A îi corespund mai mulţi membri din a doua entitate B; astfel de relaţii se mai numesc şi relaţii ierarhice.





Relaţiile de tipul m→m (mulţi la mulţi), în care unui membru din entitatea A îi corespund mai multe date din colecţia B şi mai multor date din colecţia A îi corespunde o singură dată din colecţia B.



Relaţii de tip mulţi la mulţi se mai numesc şi relaţii de tip reţea. O relaţie mulţi la mulţi se va descompune întotdeauna în două relaţii, o relaţie tip unu la mulţi şi respectiv o a doua relaţie de tip mulţi la unu prin intermediul unei entităţi de legătură.

Fiecare relaţie este prezentată în diagrama ERD ca o linie ce conectează două tabele. Cele două capete ale liniei arată *cardinalitatea maximă* a relaţiei, respectiv numărul maxim de rânduri dintr-un tabel care pot fi asociate cu un rând dat din tabelul aflat la celălalt capăt al relaţiei.

Cardinalitatea maximă poate fi:

- *unu* (caz în care linia nu are nici un simbol special la capăt) sau
- *mai multe* (caz în care linia se termină cu un simbol numit *picior de cioară (crow's foot)* - linia se împarte la capăt în trei segmente).

În apropiere de capătul liniei se află un alt simbol, care arată *cardinalitatea minimă,* adică numărul minim de rânduri dintr-un tabel care poate fi asociat cu tabelul de la celălalt capăt al relaţiei.

Cardinalitatea minimă poate fi *zero,* indicată printr-un cerc desenat pe linie, sau *unu,* indicată printr-o liniuţă care taie linia relaţiei.

De exemplu, relaţia dintre tabelele MPAA_RATING şi MOVIE din figura 1-2 este o relaţie de tip un*u-la-mai-mulţi,* ceea ce înseamnă că fiecare rând din tabelul MPAA_RATING (tabelul din partea „unu, numit şi tabel *părinte")* poate fi asociat cu mai multe rânduri din tabelul MOVIE (tabelul din partea „mai mulţi" a relaţiei, numit şi tabel *copil'"),* dar fiecare rând din tabelul MOVIE poate fi asociat cu un singur rând din tabelul MPAA_RATING.

Relaţia este logică, deoarece orice film lansat în SUA poate avea o singură categorie MPAA, dar o categorie poate fi asociată mai multor filme diferite. Este adevărat ca filmele sunt uneori „cenzurate" pentru a putea fi încadrate în diferite categorii, dar această problemă este rezolvată uşor, prin tratarea diferitelor versiuni ale aceluiaşi film ca şi cum ar fi filme diferite, la fel cum facem atunci când un film este reluat cu alţi actori. Este foarte important să ţineţi seama de aceste lucruri, deoarece bazele de date relaţionale acceptă numai relaţiile de tip unu-la-mai-mulţi sau unu-la-unu.

Cardinalitatea minimă arată dacă participarea la o anumită relaţie este obligatorie sau opţională. Toate relaţiile din fig. 1.2 sunt obligatorii în partea „unu" şi opţionale în partea „mai mulţi", aceasta fiind cea mai frecvent folosită formă de relaţie. Dacă ne uităm din nou la relaţia

dintre tabelele MPAA_RATING şi MOVIE, aceasta înseamnă că fiecare rând din tabelul MOVIE *trebuie* să aibă un rând corespondent în tabelul MPAA_RATING, dar nu este obligatoriu ca fiecare rând din tabelul MPAA_RATING să aibă asociat un rând din tabelul MOVIE. Dacă vreţi să permiteţi ca inventarul de filme al magazinului să conţină titluri care nu au asociată o categorie MPAA, liniuţa de la capătul dinspre tabelul MPAA_RATING al liniei care reprezintă relaţia cu tabelul MOVIE va fi înlocuită de un cerc. Deşi sunt relativ frecvent întâlnite cazurile în care partea „unu" a unei relaţii nu este obligatorie, este foarte neobişnuit să aveţi o relaţie în care să fie obligatorie partea „mai mulţi" a relaţiei, ceea ce ar însemna că tabelul părinte trebuie să aibă în orice moment cel puţin un „copil" în baza de date.

Relaţiile sunt implementate folosind coloane corespondente din cele două tabele participante. În diagrama ERD, coloana sau coloanele subliniate din fiecare tabel, având în dreapta notaţia „pk", reprezintă cheia primară (primary key), adică o coloană sau un set de coloane care identifică în mod unic fiecare rând dintr-un tabel.

Un tabel poate avea o singură cheie primară. Totuşi, o cheie primară poate fi compusă din mai multe coloane, dacă aceasta este calea de formare a unei chei unice. Dacă o cheie primară este folosită într-un alt tabel pentru stabilirea unei relaţii, poartă numele de cheie externă.

În fig. 1.2, observaţi coloanele cheie externă folosite în tabelul MOVIE pentru crearea relaţiilor cu tabelele MOVIE_GENRE şi MPAA_RATING şi marcate cu identificatoarele „<fk1>" şi „<fk2>" în dreapta numelui coloanei cheie externă.

Cheile primare şi cheile externe sunt blocuri de construcţie fundamentale ale modelului relaţional, deoarece stabilesc relaţii şi permit crearea legăturilor între date, atunci când este necesar. Trebuie să înţelegeţi acest concept pentru a putea înţelege cum funcţionează bazele de date relaţionale.

3. Restricţii

O restricţie este o regulă specificată pentru un obiect al bazei de date (de obicei, un tabel sau o coloană), având rolul de a limita într-un mod oarecare domeniul de valori permise pentru obiectul respectiv al bazei de date

După ce sunt specificate, restricţiile sunt impuse automat de sistemul DBMS şi nu pot fi ocolite decât dacă o persoană autorizată le dezactivează sau le şterge (le elimină). Fiecare restricţie primeşte un nume unic, astfel încât să poată fi referită în mesajele de eroare şi în comenzile folosite ulterior în baza de date. Este recomandabil ca proiectanţii bazei de date să denumească restricţiile, deoarece numele generate automat de baza de date nu sunt foarte descriptive.

Există mai multe tipuri de restricţii pentru baze de date:

- ***Restricţia NOT NULL***. Poate fi plasată pe o coloană pentru a împiedica folosirea valorilor nule. O valoare nulă (null) este o modalitate specială prin care sistemul RDBMS tratează valoarea unei coloane pentru a indica faptul că valoarea coloanei respective nu este cunoscută. O valoare nulă nu este acelaşi lucru cu un spaţiu liber, un şir vid sau valoarea zero — este o valoare specială care nu este egală cu nimic altceva.
- ***Restricţia cheie primară (primary key)***. Definită pe coloana (coloanele) cheie primară ale unui tabel pentru a garanta că valorile cheie primară sunt întotdeauna unice în întreg tabelul. Atunci când cheia primară este definită pe mai multe coloane, combinaţia valorilor acelor coloane trebuie să fie unică în tabel - o coloană care reprezintă doar o parte a cheii primare poate conţine valori duplicate în tabel. Restricţiile cheie primară sunt aproape întotdeauna implementate de RDBMS prin folosirea unui index. Indexul este un tip special de obiect al bazei de date care permite efectuarea căutărilor rapide în valorile coloanei. Atunci când în tabele sunt inserate rânduri noi, sistemul RDBMS verifică automat indexul pentru a se asigura că ***pk*** a noului rând nu este deja folosită în tabel şi, dacă se întâmplă acest lucru, respinge cererea de inserare. Căutarea în indexuri se face mult mai repede decât căutarea în tabel; ca urmare, indexarea cheii primare este esenţială pentru orice tabel, indiferent de dimensiunea acestuia, astfel încât căutarea cheilor duplicate la fiecare inserare să nu ducă la o reducere

semnificativă a performanţelor. O caracteristică suplimentară a cheilor primare este faptul că nu pot fi definite decât pe coloane pentru care a fost definită şi restricţia NOT NULL.

- ***Restricţia de unicitate (unique)***. Definită pe o coloană sau un set de coloane care trebuie să conţină valori unice în cadrul tabelului. Ca şi în cazul cheilor primare, sistemul RDBMS foloseşte aproape întotdeauna un index ca modalitate de impunere eficientă a restricţiei, Totuşi, spre deosebire de cheile primare, un tabel poate avea definite mai multe restricţii de unicitate, iar coloanele care participă la o restricţie de unicitate pot conţine (în cele mai multe sisteme RDBMS) şi valori nule.
- ***Restricţia referenţială*** (numita uneori *restricţie de integritate referenţială)*. O restricţie care impune o relaţie între două tabele dintr-o bază de date relaţională. Prin „impunere„ se înţelege că sistemul RDBMS se asigură întotdeauna, în mod automat, că fiecărei valori a cheii externe îi corespunde o valoare a cheii primare în tabelul părinte. Pe scurt, restricţia referenţială garantează că relaţia dintre cele două tabele şi valorile corespondente ale cheii primare şi cheii externe îşi păstrează logica în orice moment.
- ***Restricţia CHECK.*** Foloseşte o instrucţiune logică simplă (scrisă în SQL) pentru a valida valoarea unei coloane. Rezultatul instrucţiunii trebuie să fie o valoare logică de adevărat (true) sau fals (false), astfel încât un rezultat adevărat să permită inserarea în tabel a valorii coloanei, iar un rezultat fals să ducă la rejectarea valorii coloanei, cu mesajul de eroare corespunzător.

4. Vizualizări

O vizualizare (view) este o interogare stocată în baza de date care pune la dispoziţia utilizatorului un subset personalizat al datelor din unul sau mai multe tabele ale bazei de date. Cu alte cuvinte, o vizualizare este un tabel virtual, deoarece arată ca un tabel şi, în cele mai multe privinţe, se comportă ca un tabel, dar nu stochează date (nu este stocată decât interogarea SQL care defineşte vizualizarea). Vizualizările au mai multe funcţii utile:

- Maschează coloanele pe care utilizatorul nu este nevoie să le vadă (sau nu-i este permis să le vadă).
- Maschează rândurile pe care utilizatorul nu este nevoie să le vadă (sau nu-i este permis să le vadă).
- Maschează operaţiile complexe efectuate în baza de date, cum ar fi uniunile de tabele (respectiv combinarea coloanelor din tabele multiple într-o singură interogare a bazei de date).
- Îmbunătăţesc performanţele interogărilor (în unele sisteme RDBMS precum Microsoft SQL Server).

1.1.3. Niveluri de abstractizare a datelor

Într-un sistem informatic ce utilizează BD, organizarea datelor poate fi analizată din mai multe puncte de vedere şi pe diferite niveluri. De obicei, abordarea se face pe trei niveluri: ***intern*, *conceptual* şi *extern*** (figura 1.3).

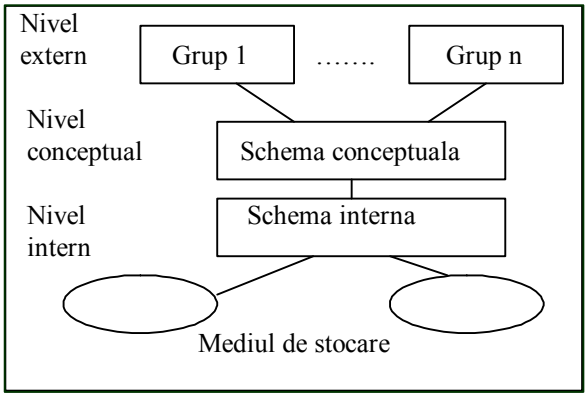


Fig. 1.3 Niveluri de abstractizare

Nivelul intern. Structura datelor este descrisă foarte detaliat, fiind accesibilă numai specialiştilor (ingineri de sistem, programatori în limbaje de asamblare sau alte limbaje apropiate

de „maşină"). Cele două părţi principale ale bazei la acest nivel sunt:

1. un set de programe care interacţionează cu sistemul de operare pentru îmbunătăţirea managementului bazei de date.
2. fişierele stocate în memoria externă a calculatorului.
   Fişierele ce conţin datele propriu-zise sunt alcătuite din articole sau înregistrări cu format comun. La acest nivel, structura BD se concretizează în schema internă.

Nivelul conceptual. Este nivelul imediat superior celui fizic, datele fiind privite prin prisma semanticii lor; interesează conţinutul lor efectiv, ca şi relaţiile care le leagă de alte date. Reprezintă primul nivel de abstractizare a lumii reale observate. Obiectivul acestui nivel îl constituie modelarea realităţii considerate, asigurându-se independenţa bazei faţă de orice restricţie tehnologică sau echipament anume. Întreaga bază este descrisă prin intermediul unui număr restrâns de structuri. Toţi utilizatorii îşi exprimă nevoile de date la nivel conceptual, prezentându-le administratorului bazei de date, acesta fiind cel care are o viziune globală necesară satisfacerii tuturor cerinţelor informaţionale. La acest nivel, structura BD se concretizează în schema conceptuală.

Nivelul extern. Este ultimul nivel de abstractizare la care poate fi descrisă o bază de date. Structurile de la nivelul conceptual sunt relativ simple, însă volumul lor poate fi deconcertant. Iar dacă la nivel conceptual baza de date este abordată în ansamblul ei, în practică un utilizator sau un grup de utilizatori lucrează numai cu o porţiune specifică a bazei, în funcţie de departamentul în care îşi desfăşoară activitatea şi de atribuţiile sale (lor). Simplificarea interacţiunii utilizatori-BD, precum şi creşterea securităţii BD sunt deziderate ale unui nivel superior de abstractizare, care este nivelul extern. Astfel, structura BD se prezintă sub diferite machete, referite uneori şi *ca sub-scheme,* scheme externe sau imagini (view-uri), în funcţie de nevoile fiecărui utilizator sau grup de utilizatori.

1.1.4. Proiectarea bazelor de date relaţionale. Etape. Normalizarea bazelor de date

Proiectarea unei baze de date este o activitate laborioasă şi necesită parcurgerea următoarelor etape:

- formularea problemei;
- analiza cerinţelor informaţionale şi definirea datelor de ieşire şi a datelor de intrare;
- definirea tabelelor, a structurii acestora şi a relaţiilor dintre tabele;
- optimizarea structurii bazei de date.

Odată ce acest proces a fost finalizat se continuă cu:

- proiectarea procedurilor tehnologice, pentru prelucrarea bazei de date;
- elaborarea programelor;
- testarea programelor;
- definitivarea documentaţiei.

Toate aceste activităţi necesită, pentru proiectele reale complexe, o muncă în echipă pe baza unei metodologii riguroase, cunoscută ca metodologia de analiză şi proiectare a sistemelor informatice. În cadrul unui sistem informatic baza de date reprezintă elementul central în jurul căruia se concentrează celelalte componente ale sistemului.

Formularea problemei presupune stabilirea obiectivelor aplicaţiei informatice care asigură actualizarea şi exploatarea bazei de date în concordanţă cu cerinţele managementului activităţii economice pentru care este proiectată baza de date. Obiectivele unei aplicaţii informatice sunt legate de asigurarea informaţională a desfăşurării proceselor decizionale specifice actului de conducere. Deci, noi trebuie să ne gândim că, prin existenţa unei baze de date, să asigurăm fondul de informaţii, într-o structură şi de o calitate corespunzătoare cu cerinţele managementului firmei. Baza de date trebuie să permită atât obţinerea unor informaţii de detaliu, elementare, cât şi calculul şi prezentarea unor indicatori sintetici, agregaţi. Dacă am lua doar două obiective: reducerea costurilor şi creşterea productivităţii muncii într-o firmă, atunci o bază de date trebuie să furnizeze informaţii despre consumul factorilor de producţie,

costurile medii şi globale, despre personalul muncitor şi producţia realizată, despre cheltuielile salariale etc. Aceste informaţii vor servi conducerii la identificarea căilor de reducere a costurilor şi adoptarea celor mai adecvate măsuri pentru reducerea acestor costuri. După aplicarea măsurilor în practică, informaţiile stocate în baza de date trebuie să permită de data aceasta şi o analiză comparată a costurilor noi cu cele vechi, de exemplu, o analiză a dinamicii costurilor pe baza indicilor statistici. Am ales acest mic exemplu didactic pentru a accentua încă o dată complexitatea procesului de proiectare a bazei de date.

Analiza cerinţelor informaţionale, pornind de la obiectivele formulate anterior, se concentrează asupra a două probleme:

- indicatorii, rapoartele, listele şi datele de ieşire care trebuie obţinute;
- datele de intrare necesare pentru obţinerea datelor de ieşire.

Acestea sunt cerinţele informaţionale care înglobează atât cerinţele pentru datele de intrare pe baza cărora se creează şi se actualizează baza de date, cât şi cerinţele pentru datele de ieşire folosite pentru urmărirea, controlul şi dirijarea activităţii economice. Datele de intrare se culeg, de regulă, din documentele primare care circulă în cadrul fluxului informaţional al firmei. Datele finale se vor integra în ansamblul de rapoarte, liste, situaţii cu rezultate pe care le furnizează sistemul informaţional compartimentelor de conducere. Pentru indicatorii incluşi în rapoartele finale, în general pentru oricare din indicatorii de ieşire, trebuie să fie foarte clar modul în care sunt obţinuţi prin prelucrarea datelor de intrare. În consecinţă, acolo unde este cazul, se precizează algoritmii de calcul, regulile de totalizare, sau alte reguli de obţinere a fiecărei coloane, sau totaluri din rapoartele finale.

Aceasta are ca punct de plecare inventarierea câmpurilor prezente în situaţiile finale şi apoi gruparea lor în tabele. Gruparea câmpurilor pe tabele se realizează prin diverse metode. Dintre aceste metode, două sunt cele mai utilizate:

- analiza concordanţei IEŞIRI – INTRĂRI;
- analiza semnificaţiei semantice a datelor.

Analiza concordanţei IEŞIRI – INTRĂRI este o tehnică specifică proiectării sistemelor informatice care identifică documentele primare din care se preiau datele de intrare folosite în calculul datelor de ieşire. Aceste documente vor constitui sursele de creare şi actualizare a tabelelor bazei de date.

Tehnica este utilă în cazul în care proiectantul bazei de date este familiarizat cu sistemul informaţional existent.

Un indicator prezent într-o situaţie finală se poate obţine astfel:

- prin preluarea directă din documentul primar, respectiv din tabelul bazei de date;
- prin aplicarea unui algoritm de calcul.

Tabelele se compun prin gruparea câmpurilor pe principiul apartenenţei acestora la anumite documente primare care circulă în cadrul sistemului informaţional.

Analiza semantică se practică atunci când proiectantul bazei de date nu are la dispoziţie un set de documente primare şi apare în cazul sistemelor informaţionale care se proiectează pentru firmele noi.

Definirea tabelelor şi relaţiilor dintre tabele este etapa următoare în proiectarea bazei de date. Analiza cerinţelor informaţionale şi a proceselor de prelucrare va conduce la identificarea datelor ce vor trebui stocate şi care vor alcătui tabelele bazei de date. O tabelă va păstra datele fie despre toate caracteristicile unei colecţii de date, fie numai pentru o parte dintre aceste caracteristici. Aici intervine spiritul analitic al proiectantului. Structura unei tabele este reprezentată de lista câmpurilor asociate tabelei împreună cu descrierea atributelor fiecărui câmp (natură, lungime, număr de zecimale etc.). În structura unui tabel se regăsesc următoarele categorii de câmpuri:

- câmpuri de identificare (chei primare şi chei condiţionate);
- câmpuri tip dată calendaristică;
- câmpuri cantitativ-valorice;
- câmpuri de legătură cu alte tabele;

- câmpuri de stare care păstrează informaţii privind ultimele operaţii de prelucrare care au fost efectuate pe înregistrările din tabel.

Relaţiile dintre tabele se caracterizează prin plasarea unor câmpuri comune în structura fiecăruia dintre tabelele aflate în relaţie directă. Pe baza acestor câmpuri, chei, fiecare sistem de gestiune a bazelor de date îşi construieşte un mecanism propriu de accesare a înregistrărilor de date. Aceste mecanisme sunt transparente pentru utilizatorul obişnuit. Totuşi este bine de reţinut că nu orice câmp poate fi folosit la stabilirea unei relaţii, a unei legături între două tabele. Numai câmpurile de tip cheie candidat, care au proprietatea de a identifica în mod unic o înregistrare dintr-o tabelă, pot fi folosite în acest scop. Cheile candidate se mai numesc şi indecşi. Operaţia prin care se construieşte sistemul de legături pentru ordonarea în vederea regăsirii înregistrărilor într-o tabelă se numeşte indexare. Prin indexare, fiecărei tabele principale de date i se va asocia o tabelă index corespunzătoare cheilor de indexare.

Optimizarea structurii bazei de date este un proces prin care se urmăreşte:

- reducerea redundanţei datelor;
- eliminarea anomaliilor de actualizare.

Reducerea redundanţei datelor până la un nivel minim şi controlat urmăreşte eliminarea duplicării inutile a unor câmpuri în mai multe tabele sau eliminarea câmpurilor obţinute prin calcul pe baza câmpurilor atomice. Un anumit nivel de redundanţă, însă, trebuie admis pentru a nu denatura realitatea reflectată de date. De exemplu, câmpul **VALOAREA_CONTRACTULUI**, se calculează după relaţia: **VALOAREA_CONTRACTULUI=CANTITATE*PRET**

Nu este recomandată eliminarea acestui câmp pe considerentul că el se obţine automat prin calcul. ***De exemplu***, în cazul în care după un anumit interval de timp, preţurile suportă o majorare globală, cum se practică foarte des, atunci automat se vor modifica şi valorile contractelor încheiate anterior datei de majorare a preţurilor ori acest lucru nu este corect, contractul odată perfectat nu-şi poate modifica preţul convenit prin negociere.

Anomaliile de actualizare se referă la anomaliile de ştergere, respectiv de modificare. De exemplu, se consideră o firmă care derulează lunar mii de contracte de aprovizionare, pentru un nomenclator foarte mare de produse agroalimentare, dar care operează numai cu câţiva furnizori. Dacă în tabela **CONTRACTE** sunt incluse alături de codul furnizorului şi denumirea furnizorului, contul său bancar şi denumirea băncii, atunci va apărea următoarea anomalie de actualizare, în cazul schimbării băncii şi a contului bancar de către furnizor. Acest lucru necesită parcurgerea tuturor înregistrărilor în care există aceste valori şi actualizarea acestora (figura 1.4).

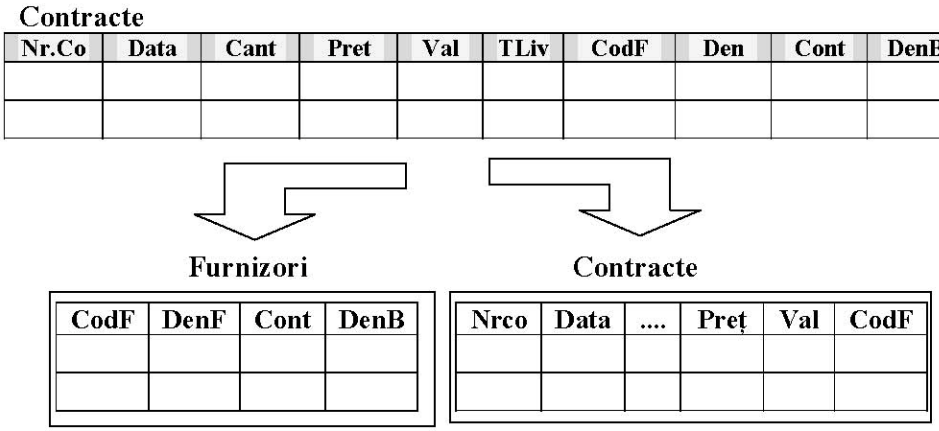


Fig. 1.4 Eliminarea anomaliilor de actualizare

Soluţia este de a crea o tabelă separat, care are în structură: codul furnizorului, denumirea, contul şi banca acestuia, în tabela **CONTRACTE** păstrându-se doar codul furnizorului ca element de legătură, ceea ce previne pierderea de informaţii prin spargerea unui tabel în două. În acest fel modificarea se realizează numai asupra unei singure înregistrări.

În 1972, Dr. E. F. Codd, părintele bazelor de date relaţionale, şi-a dat seama că tabelele

relaţionale care îndeplinesc anumite criterii pun mai puţine probleme la inserarea, actualizarea sau ştergerea datelor. Ca urmare, a pus la punct un set de reguli care trebuie respectate (organizate în trei „forme normale") şi un proces numit normalizare, care este o tehnică pentru producerea unui set de relaţii (termenul folosit de Dr. Codd pentru tabele) cu proprietăţile dorite.

Necesitatea normalizării

Figura 1.5 prezintă tabelul **MOVIE** fără normalizare, aşa cum ar arăta dacă toate informaţiile despre filme ar fi colectate într-un singur tabel. Acest exemplu va fi folosit pentru ilustrarea procesului de normalizare.

În general, numele coloanelor din tabelele relaţionale folosesc liniuţe de subliniere pentru separarea cuvintelor. În discuţia despre normalizare am eliminat aceste liniuţe din figuri, pentru a face textul mai uşor de citit.

Există trei probleme care pot apărea în tabelele fără normalizare din bazele de date relaţionale şi toate trei există şi în tabelul din figura 1.5. Scopul procesului de normalizare este de a elimina aceste probleme (anomalii) din proiectul bazei de date.

MOVIE_ID	MOVIE_GENRE_CODE	MOVIE_GENRE_DESC	LANG_CODE	MPAA_RATING_CODE	MPAA_RATING_DESC	MOVIE_TITLE	YEAR_PRODUCED	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT	RETAIL_PRICE
1	Drama	Drama	en,fr	R	under 17 requires accompanying parent or adult guardian	Mystic River	2003	01/01/2005		DVD	19.96
2	ActAd	Action and Adventure	en, fr, es	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		DVD	19.96
2	ActAd	Action and Adventure	en, fr, es	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		VHS	15.95
3	Comdy	Comedy	en	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	1/10/2005	1/30/2005	DVD	29.99
3	Comdy	Comedy	en	PG-13	Parents strongly cautioned	Something's Gotta Give	2003	2/15/2005		DVD	29.99

Fig 1.5 Tabelul MOVIE fără normalizare

Anomalia de inserare

Anomalia de inserare se referă la o situaţie în care nu puteţi insera date în baza de date din cauza unei dependenţe artificiale dintre coloanele unui tabel. ***Să presupunem*** că vreţi să adăugaţi în baza de date a magazinului de produse video un nou gen de film (în coloanele GENRE_CODE şi GENRE_DESCRIPTION) care urmează a fi folosit pentru clasificarea filmelor. Modelul din figura 1.5 nu permite acest lucru decât dacă aveţi un film care să fie plasat în categoria respectivă şi pe care va trebui să-l adăugaţi în tabelul MOVIE în acelaşi timp. Aceeaşi restricţie este valabilă şi pentru coloanele MPAA_RATING_CODE şi MPAA_RATING_DESCRIPTION. Ar fi mult mai bine dacă aţi putea adăuga noile genuri şi categorii înainte de primirea filmelor în magazin.

Anomalia de ştergere

Anomalia de ştergere este inversul anomaliei de inserare. Se referă la situaţia în care ştergerea unor date duce la pierderea neintenţionată a altor date. ***De exemplu***, dacă primul film din figura 1.5 *(Mystic River)* este singurul rând din tabelul MOVIE pentru care coloana GENRE_CODE are valoarea „Drama" şi este şters, se pierde informaţia că a existat vreodată un gen numit „Drama".

Anomalia de actualizare

Anomalia de actualizare se referă la o situaţie în care actualizarea unei singure valori necesită actualizarea mai multor rânduri. De exemplu, dacă în tabelul prezentat în figura 1.5 trebuie să modificaţi descrierea codului MPAA_RATING_CODE „R", trebuie să modificaţi şi

toate rândurile din tabel pentru filmele cu codul respectiv. Probleme similare apar şi pentru coloana GENRE_DESCRIPTION. Chiar şi coloana RETAIL_PRICE are această problemă, deoarece toate copiile aceluiaşi film (cu aceeaşi valoare MOVIE_ID) pe acelaşi mediu (DVD sau VHS) ar trebui să aibă acelaşi preţ. Un alt pericol legat de această anomalie este faptul că stocarea unor date redundante poate duce la posibilitatea de a actualiza numai o parte a copiilor respectivelor date, ceea ce ar avea ca rezultat apariţia inconsecvenţelor în baza de date.

Aplicarea procesului de normalizare

De obicei, normalizarea începe de la mijloacele de redare a datelor care sunt (sau vor fi) prezentate utilizatorilor, cum ar fi pagini web, ecrane ale aplicaţiilor, rapoarte şi aşa mai departe. Colectiv, acestea sunt numite *vizualizări de utilizator (user views)*. Poate părea ciudat la prima vedere, dar este ceva obişnuit ca proiectarea unui sistem de prelucrare a datelor să înceapă de la rezultatele pe care le va vedea utilizatorul, parcurgând apoi drumul înapoi către mijloacele folosite pentru obţinerea rezultatelor dorite.

În timpul proiectării bazei de date, procesul de normalizare este aplicat fiecărei vizualizări, iar rezultatul este un set de relaţii normalizate care pot fi apoi direct implementate ca tabele ale bazei de date relaţionale. Procesul în sine este destul de simplu, iar regulile nu sunt foarte dificile. Totuşi, stăpânirea procesului de normalizare cere timp şi exerciţiu, în special deoarece impune proiectantului să se gândească într-un mod conceptual la datele şi relaţiile pe care intenţionează sa le folosească.

În timpul normalizării, consideraţi că fiecare vizualizare este o relaţie. Cu alte cuvinte, conceptualizaţi fiecare vizualizare ca şi cum ar fi deja un tabel bidimensional - un mod de lucru pentru care aveţi nevoie de experienţă.

Reţineţi că scopul procesului de normalizare este eliminarea anomaliilor de inserare, actualizare şi ştergere. Procesul determină crearea unui număr mai mare de relaţii decât aţi avea într-un model fără normalizare. Relaţiile suplimentare sunt necesare pentru eliminarea anomaliilor, dar împărţirea datelor în mai multe relaţii face ca extragerea datelor stocate să fie puţin mai dificilă. De fapt, sacrificaţi o parte din performanţele de extragere a datelor şi din uşurinţa utilizării pentru ca operaţiile de inserare, actualizare şi ştergere să fie mai simple.

Alegerea unul identificator unic

Primul pas al procesului de normalizare constă în alegerea unui *identificator unic (unique identifier),* care este un atribut (o coloană) sau un set de atribute care identifică în mod unic fiecare rând de date dintr-o relaţie.

Identificatorul unic va deveni ulterior cheia primară a tabelului creat din relaţia normalizată. Pentru normalizare, *este obligatoriu ca* fiecare relaţie să aibă un identificator unic, în multe cazuri, puteţi găsi un atribut care identifică în mod unic datele din fiecare rând al relaţiei pe care vreţi să o normalizaţi. Atunci când nu puteţi găsi un singur atribut care să poată fi folosit ca identificator unic, este posibil să găsiţi mai multe atribute care pot fi concatenate (combinate) pentru a forma un identificator unic. Atunci când identificatoarele unice sunt formate din atribute multiple, fiecare atribut rămâne pe propria lui coloană - nu faceţi decât să definiţi un identificator unic format din mai multe coloane.

În foarte puţine cazuri, într-o relaţie nu există un set rezonabil de atribute care să poată fi folosit ca identificator unic. Atunci când se întâmplă acest lucru, trebuie să inventaţi un identificator unic, deseori cu valori atribuite secvenţial sau aleatoriu pe măsură ce noile rânduri de date sunt adăugate în tabelul bazei de date. Această tehnică este sursa unor identificatoare unice, precum numărul de asigurări sociale folosit în Statele Unite, numerele de identificare ale angajaţilor, numerele de înmatriculare ale maşinilor sau CNP.

Relaţia Movie din figura 1.5 ne pune o problemă în privinţa găsirii unui identificator unic. La prima vedere, ar părea că atributul Movie_ID este cel mai potrivit în acest scop. Totuşi, observaţi că valorile Movie_ID „2" şi „3" apar de câte două ori, aşa că, fără nici un dubiu, această valoare nu este unică. Problema este că valoarea Movie_ID identifică în mod unic fiecare titlu, dar magazinul urmăreşte separat fiecare copie a filmelor pe care le are în stoc. Cauza este faptul că magazinul se ocupă şi de închirierea filmelor şi vrea să se asigure că fiecare client

returnează exact copia pe care a închiriat-o. După inspectarea datelor folosite ca exemplu şi o scurtă discuţie cu proprietarul magazinului, ajungeţi la concluzia că în relaţia Movie nu există nici o combinaţie de atribute care să identifice în mod unic fiecare copie a unui film, aşa că inventaţi un atribut numit Copy_Number şi-l adăugaţi în relaţie. Ori de câte ori inventaţi un identificator unic (sau o parte a unui identificator) este foarte important ca toţi să înţeleagă valorile pe care le va lua acest identificator.

În acest caz, proprietarul magazinului decide ca valorile Copy_Number să reînceapă de la 1 pentru fiecare valoare Movie_ID, ceea ce înseamnă că valorile Copy_Number sunt unice numai în combinaţie cu valorile Movie_ ID. Relaţia rezultată este prezentată în figura 1.6.

Prima formă normală: eliminarea datelor repetate

O relaţie este în *prima formă normală* atunci când nu conţine atribute cu valori multiple (atribute multivaloare), adică atribute care au mai multe valori pentru acelaşi rând de date. Într-o relaţie, orice intersecţie a unui rând cu o coloană trebuie să conţină *cel mult* o valoare pentru ca relaţia să fie în prima formă normală. În figura 1.6, atributul pentru limbă (Lang. Code) conţine mai multe valori pentru unele dintre filme, aşa că-1 puteţi considera un atribut cu valori multiple.

MOVIE_ID (pk)	COPY_NUMBER (pk)	MOVIE_GENRE_CODE	MOVIE_GENRE_DESC	LANG_CODE	MPAA_RATING CODE	MPAA_RATING_DESC	MOVIE_TITLE	YEAR_PRODUCED	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT	RETAIL_PRICE
1	1	Drama	Drama	en,fr	R	under 17 requires accompanying parent or adult guardian	Mystic River	2003	01/01/2005		DVD	19.96
2	1	ActAd	Action and Adventure	en, fr, es	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		DVD	19.96

Fig. 1.6 Relaţia Movie după adăugarea atributului Copy_Number

Pentru transformarea relaţiilor ne-normalizate în prima formă normală, trebuie mutate atributele multivaloare şi grupurile repetitive în noi relaţii.

Procedura de mutare a unui atribut multivaloare sau a unui grup repetitiv într-o nouă relaţie constă în următoarele etape:

1. Creaţi o nouă relaţie, cu un nume sugestiv. Deseori, este bine să includeţi numele relaţiei originale, parţial sau în întregime, în numele noii relaţii.

2. Copiaţi identificatorul unic din prima relaţie în noua relaţie. Datele depind de acest identificator în relaţia originală, aşa că trebuie să depindă de aceeaşi cheie şi în noua relaţie. Identificatorul copiat va deveni cheie externă în noua relaţie.

3. Mutaţi grupul repetitiv sau atributul multivaloare în noua relaţie.

4. Formaţi un identificator unic în noua relaţie, adăugând atribute la identificatorul unic copiat din relaţia originală. Ca întotdeauna, asiguraţi-vă ca identificatorul unic nou format conţine numai numărul minim de atribute necesar pentru a-1 face unic. Dacă mutaţi un atribut multivaloare, care, în esenţă, este un grup repetitiv cu un singur atribut, este adăugat atributul respectiv pentru formarea identificatorului unic. Poate părea ciudat la prima vedere, dar identificatorul unic copiat din relaţia originală nu este doar o cheie externă, ci, de obicei, şi o parte a identificatorului unic (cheia primară) a noii relaţii. Acest lucru este absolut normal. De asemenea, este perfect acceptabil să avem o relaţie în care toate atributele fac parte din identificatorul unic (adică nu există atribute care să nu facă parte din cheie).

5. Opţional, puteţi înlocui cheia primară cu un singur atribut surogat pentru cheie. Dacă faceţi acest lucru, trebuie să păstraţi şi atributele care compun cheia primară naturală, formată la paşii 2 şi 4. Figura 1.7 prezintă rezultatul aducerii relaţiei din figura 1.6 la prima formă normală.

Movie

MOVIE_ID (pk)	COPY_NUMBER (pk)	MOVIE_GENRE_CODE	MOVIE_GENRE_DESC	MPAA_RATING_CODE	MPAA_RATING_DESC	MOVIE_TITLE	YEAR_PRODUCED	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT	RETAIL_PRICE
1	1	Drama	Drama	R	under 17 requires accompanying parent or adult guardian	Mystic River	2003	01/01/2005		DVD	19.96
2	1	ActAd	Action and Adventure	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		DVD	19.96
2	2	ActAd	Action and Adventure	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003	10/01/2005		VHS	15.95

Movie_Language

MOVIE_ID (fk)	LANGUAGE_CODE
1	en
1	fr
2	en
2	fr
2	es

Fig 1.7 Soluţia primei forme normale

A doua formă normală: eliminarea dependenţelor parţiale

Înainte de a explora a doua formă normală, trebuie să înţelegem conceptul *de dependenţă funcţională.*

Pentru această definiţie, vom folosi două atribute arbitrare, inteligent denumite „A" şi „B". Atributul B este *dependent funcţional* de atributul A dacă în nici un moment nu există mai mult de o valoare a atributului B asociată cu o valoare dată a atributului A.

În relaţia Movie, putem să ne dăm seama cu uşurinţa că atributul Movie_Title este dependent funcţional de atributul Movie_ID, deoarece, în orice moment, poate exista o singură valoare Movie_Title pentru o valoare Movie_ID dată. Chiar faptul că valoarea Movie_ID defineşte în mod unic valoarea Movie_Title în relaţie înseamnă că Movie_Title este dependent funcţional de Movie ID.

Se spune că o relaţie este în *a doua formă normala* dacă îndeplineşte următoarele criterii:

- Relaţia este în prima formă normală.
- Toate atributele non-cheie sunt dependente funcţional de identificatorul unic (cheia primară), *luat ca întreg.*

Aplicând aceste criterii relaţiei Movie din fig. 1.7, este clar că avem câteva probleme. Identificatorul unic este o combinaţie a atributelor Movie_ID şi Copy_Number. Totuşi, numai atributele Date_Acquired, Date_Sold, Media_Format şi Retail_Price depind de întregul identificator. Şi este logic să fie aşa. Indiferent câte copii ale unui film avem în baza de date, toate au aceleaşi valori pentru gen, categorie MPAA, titlu şi an de producţie. Cum a apărut această problemă? Ar trebui să fie clar că unele atribute descriu filmul în sine, în timp ce altele descriu, copiile pe care le deţine (sau le-a deţinut) magazinul din filmul respectiv. În esenţă, am amestecat atribute care descriu în aceeaşi relaţie două lucruri (entităţi) diferite (deşi înrudite) din lumea reală. Nici nu e de mirare că am obţinut un asemenea haos. A doua formă normală ne va ajuta să rezolvăm problemele.

A doua formă normală se aplică numai relaţiilor care au identificatoare unice concatenate (adică formate din atribute multiple). Într-o relaţie care are un singur atribut ca identificator unic,

este imposibil ca un alt atribut să depindă de o parte a identificatorului unic, deoarece acesta, fiind format dintr-un singur atribut, nu are părţi componente. **Ca urmare, orice relaţie în prima formă normală care are cheia primară formată dintr-un singur atribut este *automat în* a doua formă normală.**

Movie

MOVIE_ID (pk)	MOVIE_ GENRE_ CODE	MOVIE_ GENRE_ DESC	MPAA_ RATING_ CODE	MPAA_ RATING_ DESC	MOVIE_ TITLE	YEAR_ PRODUCED
1	Drama	Drama	R	under 17 requires accompanying parent or adult guardian	Mystic River	2003
2	ActAd	Action and Adventure	R	under 17 requires accompanying parent or adult guardian	The Last Samurai	2003

Movie_Language

MOVIE_ID (fk)	LANGUAGE_CODE
1	en
1	fr
2	en
2	fr
2	es

Movie_Copy

MOVIE_ID (pk)	COPY_NUMBER (pk)	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT	RETAIL_PRICE
1	1	01/01/2005		DVD	19.96
2	1	10/01/2005		DVD	19.96
2	2	10/01/2005		VHS	15.95
3	1	10/01/2005	30/01/2005	DVD	29.99
3	2	15/02/2005		DVD	29.99

Fig. 1.8 Soluţia pentru a doua formă normală

A treia formă normală: eliminarea dependenţelor tranzitive

Pentru a înţelege a treia formă normală, trebuie să înţelegem mai întâi conceptul de dependenţă tranzitivă.

Despre un atribut care depinde de un atribut care nu este identificator unic (cheie primară) a relaţiei se spune că *este dependent tranzitiv*.

Uitându-ne la relaţia Movie din figura 1.8, observăm că atributul Genre_Description depinde de atributul Genre_Code, iar MPAA_Rating_Description depinde de MPAA_Rating_Code. Pericolul păstrării acestor descrieri în relaţia Movie este faptul că, în final, cele două atribute ajung să depindă de înregistrarea unui film, ceea ce duce la toate cele trei anomalii de date prezentate mai devreme în acest capitol.

Se spune că o relaţie este în *a treia formă normală* dacă îndeplineşte următoarele două criterii:

- Relaţia este în a doua formă normală.
- Nu există dependenţe tranzitive (cu alte cuvinte, toate atributele non-cheie depind *numai* de identificatorul unic).

Pentru a aduce la a treia formă normală o relaţie aflată în a doua formă normală, mutăm atributele dependente tranzitiv în relaţii în care depind numai de cheia primară

Avem grijă să lăsăm atributul de care depind acestea în relaţia originală, cu rolul de cheie externă. Va trebui apoi să reconstruim vizualizarea originală printr-o uniune. Ca efect secundar, toate atributele uşor de calculat sunt eliminate ca încălcări ale criteriilor celei de-a treia forme normale.

De exemplu, într-o bază de date pentru vânzări, Suma Totală este obţinută înmulţind Cantitatea Cumpărată cu Preţul Unitar; aşa cum se observă cu uşurinţă, Suma Totală este dependentă de Cantitatea Cumpărată şi de Preţul Unitar. Presupunând că toate cele trei atribute sunt dependente de identificatorul unic al relaţiei care le conţine, este uşor de văzut că Suma Totală (rezultatul calculat) este, de fapt, *dependentă tranzitiv* de celelalte două atribute. Figura 1.9 conţine soluţia în a treia formă normală.

Movie

MOVIE_ID (pk)	MOVIE_GENRE_ CODE	MPAA_ RATING_ CODE	MOVIE_TITLE	RETAIL_ PRICE_ VHS	RETAIL_ PRICE_ DVD	YEAR_ PRODUCED
1	Drama	R	Mystic River	58.97	19.96	2003
2	ActAd	R	The Last Samurai	15.95	19.96	2003

Movie_Language

MOVIE_ID (fk)	LANGUAGE_CODE
1	en
1	fr
2	en
2	fr
2	es

Movie_Copy

MOVIE_ID (pk)	COPY_NUMBER (pk)	DATA_ACQUIRED	DATE_SOLD	MEDIA_FORMAT
1	1	01/01/2005		DVD
2	1	10/01/2005		DVD
2	2	10/01/2005		VHS
3	1	10/01/2005	30/01/2005	DVD
3	2	15/02/2005		DVD

MPAA_Rating

MPAA_RATING_CODE (pk)	MPAA_RATING_DESCRIPTION
PG-13	parents strongly cautioned
R	under 17 requires accompanying parent or adult guardian

Movie_Genre

MOVIE_GENRE_CODE (pk)	MOVIE_GENRE_DESCRIPTION
ActAd	Action and Adventure
Comedy	Comedy
Drama	Drama

Fig 1.9 Soluţia pentru a treia formă normală

Prezentare generală a bazei de date pentru un magazin de produse video

Toate exemplele prezentate în cadrul cursului folosesc ca suport o bază de date pentru un magazin de produse video.

Figura 1.10 prezintă diagrama de relaţii a entităţilor (ERD - Entity Relationship Diagram) pentru întreaga bază de date a magazinului de produse video.

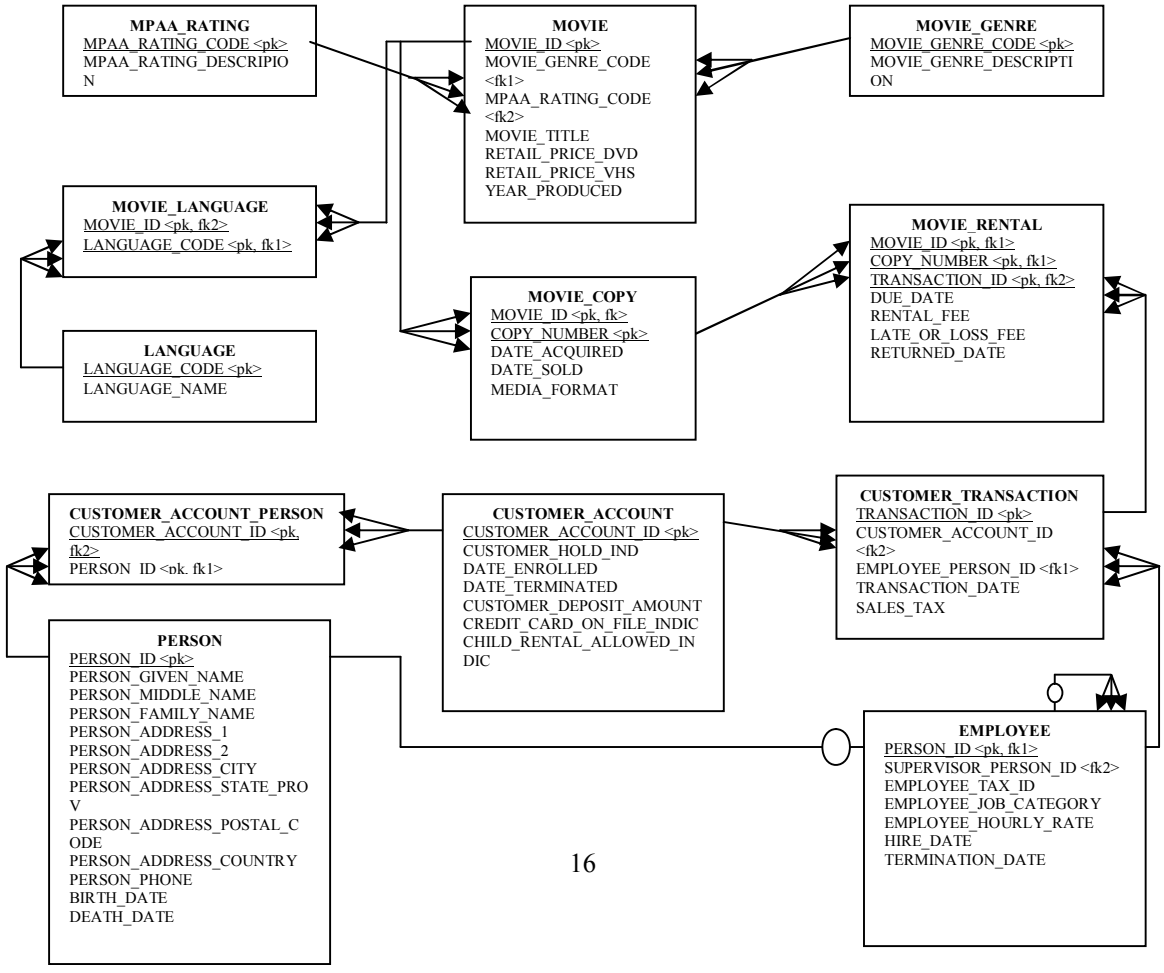


Fig. 1.10 Diagrama ERD a magazinului de produse video

1.2. CONCEPTE SQL

SQL a devenit limbajul universal pentru bazele de date relaţionale şi este acceptat de aproape toate sistemele RDBMS moderne. Fără îndoială, acceptarea pe scară largă este rezultatul timpului şi eforturilor depuse pentru dezvoltarea caracteristicilor limbajului şi a standardelor, crescând nivelul de portabilitate a codului SQL între diferitele produse RDBMS.

SQL (Structured Query Language - limbaj structurat de interogare) este un limbaj standard folosit pentru comunicarea cu bazele de date. Numele limbajului poale fi pronunţat pe litere (es-q-el) sau la fel ca şi cuvântul englezesc „sequel". O *interogare (query)* este o simplă cerere transmisă către baza de date, la care aceasta răspunde într-o anumită formă. SQL este limbajul folosit cel mai frecvent pentru interogarea bazelor de date. SQL este considerat un limbaj *neprocedural* sau *declarativ* ceea ce înseamnă că-i spuneţi calculatorului ce rezultate vreţi, fără să-i spuneţi cum să le obţină.

De exemplu, dacă vreţi să obţineţi media numerelor de pe o coloană, folosiţi funcţia AVG. Nu este nevoie să număraţi valorile din coloană şi să împărţiţi suma acestora la numărul obţinut - procesorul limbajului SQL din DBMS se ocupă de toate aceste lucruri în locul dumneavoastră.

Este important să înţelegem că SQL nu este un limbaj procedural, ca C, Pascal, Basic, FORTRAN, COBOL sau Ada. Un limbaj *procedural* foloseşte o serie de instrucţiuni executate secvenţial. De asemenea, limbajele procedurale includ instrucţiuni care pot modifica secvenţa de execuţie, prin ramificarea la alte porţiuni ale procedurii sau prin parcurgerea ciclică a unui set de instrucţiuni din procedură. Mulţi producători de sisteme RDBMS oferă extensii procedurale ale limbajului SQL de bază, cum ar fi Oracle PL/SQL {Procedural Language/SQL) sau Microsoft Transact-SQL, dar reţineţi că acestea sunt extensii SQL care formează noi limbaje - codul SQL pe care-1 conţin rămâne neprocedural. De asemenea, SQL nu trebuie confundat cu limbajele orientate spre obiecte, precum Java şi C++.

Simplu spus, SQL este un limbaj pentru gestionarea şi întreţinerea bazelor de date relaţionale, nu un limbaj potrivit pentru programarea generală a aplicaţiilor, cum ar fi sistemele de prelucrare a comenzilor sau a plăţilor.

SQL este deseori folosit în combinaţie cu limbajele procedurale sau orientate spre obiecte menţionate anterior pentru a manipula stocarea şi extragerea datelor, folosind instrucţiuni din limbajul de programare cu destinaţie generală pentru alte sarcini de programare, precum prezentarea datelor pe o pagină web sau furnizarea răspunsurilor la informaţiile introduse de utilizatori de la tastatură sau mouse. Atunci când este necesară interacţionarea cu baza de date, instrucţiunile din limbajul procedural formează instrucţiunea SQL, o transmit către RDBMS în vederea prelucrării, primesc rezultatele returnate de RDBMS şi le prelucrează într-un mod corespunzător.

1.2.1. Conectarea la baza de date

Atunci când folosiţi limbajul SQL pe un calculator personal, cu o copie personală a unui sistem DBMS, precum Microsoft Access sau Oracle Personal Edition, toate componentele bazei de date rulează pe acelaşi sistem de calcul. Totuşi, acest aranjament nu este potrivit pentru bazele de date care trebuie să fie folosite în comun de mai mulţi utilizatori.

Ca urmare, sunt mult mai frecvent întâlnite situaţiile în care baza de date este instalată într-o arhitectură client/server, ca în figura 2.1.

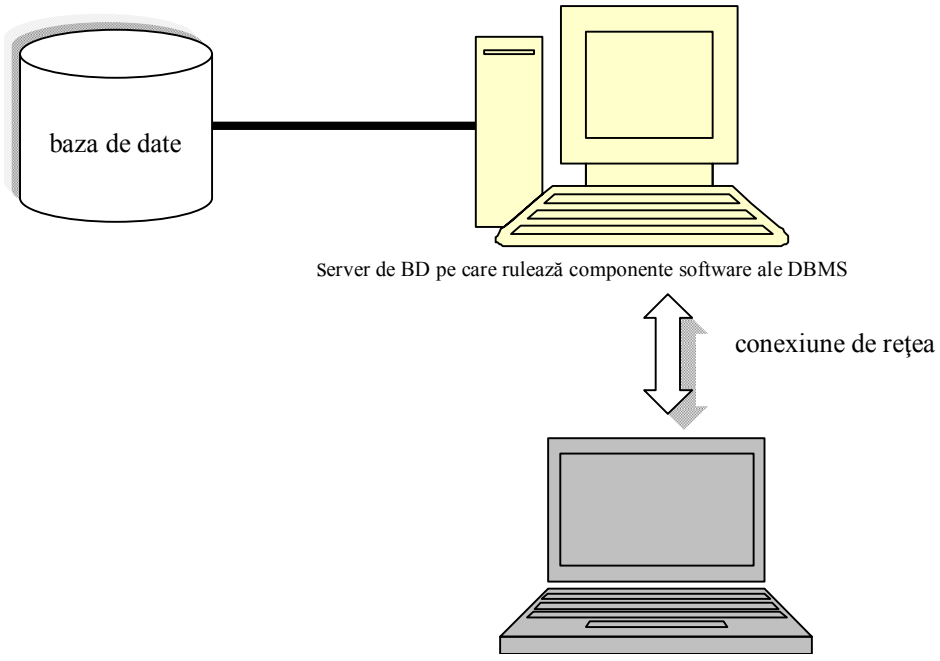


Fig. 2.1 Conexiunea clientului SQL cu baza de date

Într-o arhitectură client/server:

- Sistemul RDBMS rulează pe un *server,* care este un sistem de calcul partajat. Pentru scopurile acestei definiţii un sistem mainframe poate fi considerat un server de dimensiuni mari.
- Fişierele care compun baza de date din punct de vedere fizic sunt stocate pe discuri conectate la serverul de baze de date.
- Utilizatorii care au acces la baza de date folosesc staţii de lucru, numite *clienţi.* Clientul trebuie să aibă o conexiune de reţea la baza de date, care poate fi o reţea privată, instalată acasă sau la birou, ori o reţea publica, precum Internet.
- Componentele software furnizate de producătorul DBMS rulează pe staţiile de lucru alte clienţilor pentru a oferi utilizatorilor posibilitatea să introducă instrucţiuni SQL, să le transmită sistemului DBMS în vederea prelucrării şi să vadă rezultatele returnate de DBMS. În general, acest software se numeşte client SQL.

De reţinut că: nimic nu ne opreşte să instalăm clientul SQL pe acelaşi calculator cu sistemul DBMS. De fapt, mulţi dezvoltatori care utilizează sisteme DBMS precum MySQL, Microsoft SQL Server şi Oracle fac în mod obişnuit acest lucru, deoarece este foarte convenabil să aibă întregul mediu de dezvoltare pe un singur calculator, cum ar fi un laptop.

Totuşi, în momentul în care este necesar accesul partajat al mai multor utilizatori, este mult mai convenabil şi mai eficient să aveţi o singură copie a sistemului DBMS pe un server partajat şi să aveţi numai clientul SQL instalat pe staţia de lucru a fiecărui client.

În funcţie de interfaţa cu utilizatorul de pe staţia de lucru client, clienţii SQL sunt clasificaţi în trei categorii:

- ***în linia de comandă:*** o interfaţă *în linia de comandă se* bazează exclusiv pe intrări şi ieşiri de tip text, cu comenzile introduse de la tastatură şi răspunsurile afişate ca mesaje de tip text. Principalul avantaj al interfeţelor în linia de comandă este că pot fi rulate pe aproape orice sistem de operare.
- ***grafici:*** o *interfaţă grafică cu utilizatorul (GUI - graphical user interface)* rulează sub un tip

oarecare de sistem bazat pe ferestre, cum ar fi X Window System, Mac OS sau Microsoft Windows, şi afişează datele sau opţiunile comenzilor folosind elemente grafice, precum pictograme, butoane şi casete de dialog.

- ***bazaţi pe web:*** o *interfaţă bazată pe web* rulează pe serverul de baze de date, folosind un browser web de pe staţia de lucru client pentru a interacţiona cu utilizatorul bazei de date. Din punct de vedere tehnic, un client SQL bazat pe web nici nu este o aplicaţie client, deoarece nu există nici o componentă software specifică producătorului DBMS rulată pe staţia de lucra a clientului. Totuşi, aproape întotdeauna există componente furnizate de producătorul DBMS care sunt descărcate în fundal de browser-ul web pentru a asista în procesul de reprezentare grafică a formularelor web folosite pentru introducerea instrucţiunilor SQL şi afişarea rezultatelor.

1.2.2. Convenţii de sintaxă SQL

Această secţiune prezintă convenţiile generale de sintaxă folosite pentru construirea instrucţiunilor SQL. Convenţiile de sintaxă SQL sunt mai uşor de înţeles folosind un exemplu simplu. Instrucţiunea de mai jos returnează valorile Movie_ID şi Movie_Title pentru toate filmele din magazinul de produse video pentru care categoria MPAA este „PG":

```
SELECT MOVIE_ID, MOVIE_TITLE FROM MOVIE WHERE MPAA_RATING
_CODE = 'PG';
```

Convenţiile de bază sunt următoarele:

- Fiecare instrucţiune începe cu o comandă, de obicei sub forma unui singur cuvânt, care aproape întotdeauna este un verb (în limba engleză) care descrie o acţiune, în acest exemplu, instrucţiunea începe cu comanda SELECT.
- Fiecare instrucţiune se termină cu un delimitator, care este, de obicei, un caracter punct şi virgulă (;).
- Instrucţiunile sunt construite într-o manieră similară cu propoziţiile din limba engleză, cu unul sau mai multe spaţii pentru separarea elementelor de limbaj. Un *element de limbaj,* asemănător cu un cuvânt dintr-o propoziţie, poate fi un cuvânt cheie (SELECT, FROM, WHERE), numele unui obiect al bazei de date (MOVIE, MOVIE_ID, MOVIE_TITLE), un operator (=) sau o constantă ('PG') care apare într-o instrucţiune.
- Instrucţiunile sunt scrise într-o formă liberă, ceea ce înseamnă că nu există reguli stricte privind poziţia elementelor de limbaj pe o linie sau locul în care se poate face trecerea la o linie nouă. Totuşi, în general nu este o idee bună să împărţiţi un element de limbaj pe mai multe linii.
- Instrucţiunile sunt organizate într-o serie de clauze şi, de obicei, clauzele trebuie să apară într-o anumită ordine atunci când sunt folosite (multe clauze sunt opţionale). În exemplul nostru, există trei clauze, fiecare începând cu un cuvânt cheie (SELECT, FROM, WHERE).
- Elementele de limbaj SQL pot fi scrise cu litere mari, cu litere mici sau în combinaţii. Aceasta nu înseamnă că datele nu pot conţine litere mici, ci că numele obiectelor din baza de date (tabele, coloane etc.) şi comenzile trebuie să fie scrise cu litere mari.
- Virgulele sunt folosite pentru separarea articolelor dintr-o listă. În exemplul nostru, numele a două coloane sunt specificate într-o listă separată prin virgule (MOVIE_ID, MOVIE_TITLE). Spaţiile care urmează după virgule sunt opţionale puteţi adăuga orice număr de spaţii, inclusiv zero.
- Şirurile de caractere care apar în instrucţiunile SQL trebuie să fie încadrate cu apostrofuri (unele implementări SQL permit şi folosirea ghilimelelor). Constantele numerice nu sunt niciodată încadrate cu apostrofuri. Dacă în şirul de caractere trebuie să apară un caracter apostrof sunt inserate două apostrofuri unul lângă celălalt. De exemplu, dacă vreţi să găsiţi în baza de date un film numit *Sophie's Choice,* veţi scrie clauza WHERE astfel;
    - WHERE MOVIE_NAME = *'Sophie' 's Choice'*

- Numele obiectelor bazei de date sunt formate folosind *numai* litere, cifre şi liniuţe de subliniere. Caracterul *underscore* (liniuţa de subliniere) este folosit, de obicei, ca separator între cuvinte, pentru îmbunătăţirea lizibilităţii.
- În fiecare implementare SQL este definit un set *de cuvinte rezervate*$_t$ care sunt cuvinte cu o semnificaţie specifică pentru procesorul SQL al sistemului DBMS şi, ca urmare, nu trebuie folosite într-un alt context — de exemplu ca nume pentru obiectele bazei de date.
- Un comentariu pe o singură linie începe cu două liniuţe de despărţire (--), Cele două liniuţe se pot afla la începutul unei linii, ceea ce înseamnă că întreaga linie este considerată comentariu, sau oriunde în cadrul liniei, caz în care restul liniei, până la sfârşit, este considerat comentariu. De exemplu:

  --Acesta este un comentariu pe o singură linie în SQL.
- Un comentariu pe mai multe linii începe cu combinaţia dintre o diagonală la dreapta (slash) şi un asterisc (/*) şi continuă până la întâlnirea combinaţiei inverse (*/). Un exemplu de comentariu pe mai multe linii:

  / *Acesta este un comentariu pe mai multe linii.*
  Continuă până la întâlnirea combinaţiei de caractere care marchează sfârşitul comentariului. **/*

Instrucţiunile SQL sunt împărţite în categorii, după funcţiile pe care le îndeplinesc. Categoriile de instrucţiuni, descrise în secţiunile următoare, sunt:

- Limbajul de definire a datelor (DDL - Data Definition Language)
- Limbajul de interogare a datelor (DQL - Data Query Language)
- Limbajul de manipulare a datelor (DML - Data Manipulation Language)
- Limbajul pentru controlul datelor (DCL - Data Control Language)
- Comenzile pentru controlul tranzacţiilor (Transaction Control Commands)

Limbajul de definire a datelor (DDL): include instrucţiuni SQL care permit utilizatorului bazei de date să creeze şi să modifice structura obiectelor bazei de date, cum ar fi tabele, vizualizări şi indexuri. Instrucţiunile SQL care folosesc comenzile CREATE, ALTER şi DROP sunt considerate parte a DDL.

Este important să înţelegeţi că instrucţiunile DDL afectează containerele care stochează datele în baza de dale, nu datele propriu-zise. Ca urmare, există instrucţiuni DDL pentru crearea, ştergerea şi modificarea tabelelor, dar nici una dintre aceste instrucţiuni nu oferă posibilitatea de a crea sau modifica rânduri de date din tabelele respective.

Limbajul de interogare a datelor (DQL): include instrucţiuni SQL care permit obţinerea datelor din baza de date. Deşi reprezintă o parte foarte importantă a limbajului SQL, DQL este format din instrucţiuni care folosesc o singură comandă: SELECT.

Limbajul de manipulare a datelor (DML): include instrucţiuni SQL care permit utilizatorului bazei de date să adauge date în baza de date (sub forma rândurilor din tabele), să şteargă date şi să modifice datele existente în baza de date. Instrucţiunile SQL care folosesc comenzile INSERT, UPDATE şi DELETE sunt considerate parte a DML.

Limbajul pentru controlul datelor (DCL): include instrucţiuni SQL care permit administratorilor să controleze accesul la datele din baza de date şi folosirea diferitelor privilegii ale sistemului DBMS, cum ar fi privilegiul de oprire şi pornire a bazei de date. Instrucţiunile SQL care folosesc comenzile GRANT şi REVOKE sunt considerate parte a DCL.

Comenzile pentru controlul tranzacţiilor: *o tranzacţie în baza de date* este un set de comenzi pe care utilizatorul bazei de date vrea să le trateze ca pe o unitate funcţională de tip ‘totul sau nimic’, înţelegând prin aceasta că întreaga tranzacţie trebuie să reuşească sau să eşueze. Comenzile pentru controlul tranzacţiilor (Transaction Control Commands) nu respectă cu exactitate sintaxa instrucţiunilor SQL, dar afectează puternic comportamentul instrucţiunilor SQL incluse în tranzacţii.

1.3. DEFINIREA ŞI INTEROGAREA DATELOR FOLOSIND LIMBAJUL SQL

1.3.1. Instrucţiuni DDL (Data Definition Language)

Instrucţiunile DDL (Data Definition Language) definesc obiectele bazei de date, dar nu inserează şi nu actualizează date în obiectele respective (această funcţie fiind îndeplinită de instrucţiunile DML).

În SQL, există trei comenzi de bază pentru instrucţiunile DDL:

- **CREATE** - Creează în baza de date un nou obiect, de tipul specificat în instrucţiune. Deoarece sintaxa este diferită, vom prezenta separat instrucţiunile CREATE DATABASE, CREATE TABLE, CREATE INDEX şi CREATE VIEW.
- **ALTER** - Modifică definiţia unui obiect existent în baza de date, de tipul specificat în instrucţiune.
- **DROP** - Şterge (distruge) un obiect existent în baza de date, de tipul specificat în instrucţiune.

Instrucţiunea CREATE DATABASE

Sintaxa generală pentru instrucţiunea CREATE DATABASE este

CREATE DATABASE *nume_baza_de_date [opţiuni specifice producătorului]*

Instrucţiunea CREATE TABLE

CREATE TABLE este una din instrucţiunile fundamentale din SQL. Modelul relaţional cere ca toate datele stocate să fie ancorate într-un tabel, aşa că posibilitatea de a stoca orice într-o bază de date începe întotdeauna cu crearea unui tabel. Sintaxa de bază pentru instrucţiunea CREATE TABLE este

CREATE TABLE *nume_ tabel* (<definiţie_coloana> [,<definiţie_coloană> ...])

[, <restricţie_tabel> ...] ;

Fiecare instrucţiune include numele tabelului şi o listă cu una sau mai multe definiţii de coloane, separate prin virgule şi încadrate între paranteze. Numele tabelului trebuie să fie unic în baza de. Fiecare tabel trebuie să aibă cel puţin o coloană.

Definirea coloanelor în SQL DDL

Sintaxa de bază folosită pentru definirea coloanelor unui tabel este:

<definiţie_coloana> :

nume_coloană tip_de_date [DEFAULT *expresie*] [NULL | NOT NULL] [<restricţie_coloană>]

Restricţiile coloanelor

Restricţiile unei coloane limitează (constrâng) într-un mod oarecare valorile ce pot fi stocate într-o coloană a unui tabel.

Din punct de vedere tehnic, clauzele DEFAULT şi NULL sau NOTNULL sunt forme speciale de restricţii, dar acestea nu sunt implementate întotdeauna în acelaşi mod în toate sistemele DBMS. Restricţia unei coloane se poate referi la o singură coloană, dar există o modalitate simplă de ocolire, deoarece orice restricţie de coloană poate fi rescrisă ca restricţie de tabel. Restricţiile coloanelor pot avea oricare dintre următoarele forme:

Clauza **DEFAULT**

O expresie care este aplicată coloanei atunci când în tabel este inserat un nou rând, care nu conţine o valoare explicită pentru coloana respectivă. Expresia poate fi orice expresie validă care poate fi interpretată de SQL, cum ar fi o constantă, o funcţie SQL sau o altă sintaxă care, în urma evaluării de către motorul SQL din DBMS, produce o valoare corespunzătoare pentru coloana respectivă.

Ca ***exemplu***, observaţi clauza DEFAULT 'N' pentru coloana CUSTOMER_HOLD_INDIC din tabelul CUSTOMER_ACCOUNT. Prin specificarea acestei valori prestabilite vă asiguraţi că orice cont de client nou creat va primi întotdeauna valoarea 'N'

(contul nu este blocat) dacă instrucţiunea care inserează noul rând nu furnizează o valoare pentru această coloană sau foloseşte cuvântul cheie DEFAULT pentru valoarea coloanei. O altă utilizare obişnuită a clauzei DEFAULT este inserarea unor date, cum ar fi stocarea datei curente pentru efectuarea unei tranzacţii, la inserarea noilor rânduri în baza de date. Iată sintaxa SQL şi un exemplu de coloană cu o clauză DEFAULT:

[DEFAULT *expresie*]

Exemplu:

CUSTOMER_HOLD_INDIC CHAR(l) DEFAULT 'N' NOT NULL

Restricţia **NULL | NOT NULL**

Specificarea cuvântului cheie NULL permite stocarea valorilor nule într-o coloană, în timp ce NOT NULL nu permite stocarea valorilor nule în coloana respectivă. Iată sintaxa SQL şi câteva exemple:

NULL | NOT NULL

Exemple:

DATE_ENROLLED DATE NOT NULL

DATE_TERMINATED DATE NULL

Restricţia **CHECK**

O restricţie de verificare (check) poate fi folosită pentru impunerea unei reguli care poate fi aplicată unei singure coloane a unui tabel. Condiţia inclusă în restricţie trebuie să fie îndeplinită ori de câte ori datele din coloana respectivă a tabelului sunt modificate - în caz contrar, sistemul DBMS va respinge modificarea şi va afişa un mesaj de eroare. Condiţia din restricţia CHECK a unei coloane nu poate referi nici o altă coloană a tabelului.

Iată sintaxa restricţiei CHECK şi un exemplu:

[CONSTRAINT nume_restrictie] CHECK (conditie)

Exemplu:

CREDIT_CARD_ON_FILE_INDIC CHAR(l) NOT NULL

CHECK (CREDIT_CARD_ON_FILE_INDIC IN 'Y', 'N')

Restricţia **UNIQUE**

O restricţie UNIQUE impusă asupra unei coloane garantează unicitatea valorilor din coloana respectivă a tabelului, de obicei cu ajutorul unui index creat automat de DBMS.

Iată sintaxa restricţiei de unicitate şi un exemplu de utilizare:

[CONSTRAINT *nume_restricţie*] UNIQUE

Exemplu:

CUSTOMER_ACCOUNT_ID INTEGER NOT NULL UNIQUE

Restricţia **PRIMARY KEY**

O restricţie de cheie primară (PRIMARY KEY) impusă asupra unei coloane declară coloana respectivă ca fiind cheia primară a tabelului, ceea ce înseamnă că în coloana respectivă nu pot exista valori nule, iar valorile trebuie să fie unice în cadrul tabelului. Ca şi în cazul restricţiilor de unicitate, majoritatea sistemelor DBMS impun restricţia de cheie primară cu ajutorul unui index creat automat.

Iată sintaxa restricţiei de cheie primară şi un exemplu de utilizare:

[CONSTRAINT *nume_restricţie*] PRIMARY KEY

Exemplu:

CUSTOMER_ACCOUNT_ID INTEGER NOT NULL PRIMARY KEY

Restricţia referenţială **(FOREIGN KEY)**

O restricţie referenţială impusă asupra unei coloane (numită uneori şi restricţie de cheie externă) defineşte relaţia dintre o cheie externă şi o cheie primară, astfel încât sistemul DBMS să poată garanta că valoarea, dacă nu este nulă, referă întotdeauna valoarea unei chei primare existente. Iată sintaxa restricţiei referenţiale:

[CONSTRAINT *nume_restricţie*]

REFERENCES *nume_ tabel* (*nume_coloană)*
[ON DELETE CASCADE | ON DELETE SET NULL]
Exemplu:
MPAA_RATING_CODE CHAR(5) NOT NULL
REFERENCES MPAA_RATING (MPAA_RATING_CODE)

Clauza opţională ON DELETE spune sistemului DBMS ce să facă atunci când este şters rândul referit din tabelul părinte (rândul care conţine cheia primară corespondentă), cu opţiunea de a şterge toate rândurile care conţin cheia externă (CASCADE) sau de a insera valori nule pentru toate cheile externe (SET NULL). Reţineţi că majoritatea sistemelor DBMS, dar nu toate, acceptă clauza ON DELETE.

Restricţiile tabelelor

Restricţia unei coloane poate fi rescrisă şi ca restricţie a întregului tabel, astfel încât clauza care defineşte restricţia să apară în instrucţiunea CREATE TABLE după definiţiile tuturor coloanelor, nu după definiţia unei coloane. Principalul avantaj al restricţiilor la nivelul tabelului este că pot referi mai multe coloane. Semnificaţia fiecărei restricţii a fost deja discutată (în secţiunea despre restricţiile coloanelor) aşa că aici vom prezenta numai sintaxa generală şi câteva exemple.

Toate exemplele folosesc tabelul CUSTOMER_ACCOUNT, dar unele au fost modificate, astfel încât să ilustreze ideile principale prezentate.

Restricţia **CHECK**
[CONSTRAINT *nume_restricţie]* CHECK *(condiţie)*
Exemplu:
CONSTRAINT CK_CUSTOMER_DEPOSIT_AMOUNT
CHECK (CUSTOMER_DEPOSIT_AMOUNT >= 0 OR CUSTOMER_DEPOSIT_AMOUNT
IS NULL)

Restricţia din exemplul de mai sus împiedică stocarea unei valori negative în coloana CUSTOMER_DEPOSIT_AMOUNT. Observaţi operatorul SAU, folosit pentru a permite stocarea valorilor nule în coloană. Dacă nu ar fi fost inclusă această condiţie, coloana nu ar fi acceptat valori nule, deoarece o valoare nulă nu este mai mare sau egală cu zero.

Restricţia **UNIQUE**
[CONSTRAINT *nume_restricţie]* UNIQUE*(nume_coloana[, nume_coloana ...,])*
Exemplu:
CONSTRAINT UK_CUST_ACCT_DATE_ENROLLED
UNIQUE (CUSTOMER_ACCOUNT_ID, DATE_ENROLLED)

Conform acestei restricţii, combinaţia de coloane CUSTOMER_ACCOUNT_ID şi DATE_ENROLLED trebuie să fie unică în rândurile din tabelul CUSTOMER_ACCOUNT.

În acest exemplu, coloana CUSTOMER_ACCOUNT_ID este oricum unică, aşa că impunerea noii restricţii nu prea are sens, dar am inclus-o aici pentru a ilustra folosirea unei restricţii de unicitate bazate pe mai multe coloane.

Restricţia **PRIMARY KEY**
[CONSTRAINT *nume_restricţie]* PRIMARY KEY*(nume_coloana[,nume_coloana..,])*
Exemplu:
CONSTRAINT PK_CUSTOMER_ACCOUNT
PRIMARY KEY (CUSTOMER_ACCOUNT_ID)

Restricţia de mai sus este chiar definiţia cheii primare din tabelul CUSTOMER_ACCOUNT folosit ca exemplu în această secţiune, dar în acest caz am denumit explicit restricţia.

Restricţia referenţială **(FOREIGN KEY)**
[CONSTRAINT *nume_restricţie]*
FOREIGN KEY *(nume coloană* [*, nume_coloană..,*]*)* REFERENCES

nume_tabel(nume_coloană [, nume_coloană...]) [ON DELETE CASCADE | ON DELETE SET NULL)

Observaţi că, spre deosebire de forma pentru coloane a restricţiei referenţiale, aceasta poate referi coloane multiple. Aşa cum a fost proiectat, tabelul CUSTOMER_ACCOUNT nu are coloane chei externe, dar să ne gândim la un alt model.

Unii proiectanţi de baze de date nu sunt de acord cu folosirea restricţiilor CHECK pentru a controla valorile din coloane, deoarece adăugarea sau eliminarea valorilor implică modificarea proiectului bazei de date. Să presupunem, de exemplu, că la o îmbunătăţire ulterioară a bazei de date pentru magazinul de produse video trebuie să permitem stocarea unei noi valori, 'E' (de la „exceptat"), în coloana CREDIT_CARD_ON_FILE_INDIC. Puteţi modifica restricţia CHECK pentru a permite stocarea noii valori în tabel, dar dacă aţi fi pus toate valorile şi descrierile într-un tabel separat (deseori numit tabel „de coduri", „referinţă" sau „de căutare"), ar fi fost suficient să adăugaţi noul cod în tabelul respectiv, fără nici o modificare asupra proiectului bazei de date. Tocmai pentru obţinerea acestei flexibilităţi, baza de date a magazinului de produse video conţine tabele pentru informaţii precum MPAA_RATING_CODE - dacă asociaţia MPAA îşi schimbă sistemul de clasificare, este suficient să modificaţi datele din tabelul de coduri. De asemenea, tabelele de coduri sunt o sursă elegantă pentru popularea cu valori a listelor derulante din componente de aplicaţie, precum formularele din paginile web.

Presupunând că a fost creat un tabel numit CARD_ON_FILE_TYPE, cu o cheie primară numită CARD_ON_FILE_CODE, iată cum arată restricţia referenţială care defineşte coloana CREDIT_CARD_ON_FILE_INDIC ca fiind cheie externă:

CONSTRAINT FK_CARD_ON_FILE_INDIC FOREIGN KEY (CREDIT_CARD_ON_FILE_INDIC) REFERENCES CARD_ON_FILE_TYPE (CARD_ON_FILE_CODE)

Aşa cum puteţi vedea, nu întotdeauna există un singur mod „corect" de proiectare a unei baze de date, ci mai multe soluţii din care puteţi alege. Altfel spus, proiectarea bazelor de date nu este o ştiinţă exactă. În general, este recomandabil să daţi cheii externe acelaşi nume ca şi cheii primare, dar aşa cum puteţi vedea din acest exemplu, SQL vă permite să folosiţi un alt nume, dacă aşa vreţi sau aşa trebuie.

Instrucţiunea CREATE INDEX

CREATE [UNIOUE] INDEX *nume_index* ON *nume_tabel (nume_coloană [,nume coloană* [ASC | DESC]...]);

- Cuvântul cheie opţional UNIQUE defineşte indexul ca unic, însemnând că nu pot exista două rânduri din tabel cu exact aceeaşi combinaţie de valori în coloanele specificate.
- Cuvântul cheie opţional ASC creează indexul în ordine crescătoare, în timp ce DESC creează indexul în ordine descrescătoare. Dacă nu este specificată nici una dintre cele două opţiuni, ordinea prestabilită este crescătoare.
- Un index trebuie să aibă cel puţin o coloană, dar, practic, nu există o limită superioară a numărului de coloane.

Indexurile sunt instrumente puternice, deoarece permit sistemului DBMS să găsească datele mult mai repede, aşa cum indexul unei cărţi vă permite să găsiţi rapid ceea ce vă interesează. Mai mult, indexurile pe coloanele cheilor externe cresc mult performanţele la unirea tabelelor. Totuşi, indexurile ocupă spaţiu de stocare şi trebuie să fie întreţinute de fiecare dată când se modifică o valoare de pe o coloană referită de un index, trebuie să fie modificat şi indexul corespunzător. Sistemul DBMS întreţine automat indexul, dar activitatea de întreţinere consumă din resursele calculatorului.

Instrucţiunea CREATE VIEW

Vizualizările oferă mari avantaje utilizatorilor unei baze de date, deoarece permit ajustarea datelor afişate în funcţie de cerinţele individuale şi maschează operaţiile complexe. Atunci când sunt create corect, vizualizările implică suprasarcini minime şi nu stochează date.

În esenţă, o vizualizare este o interogare SQL stocată, care poate fi referită de instrucţiunile SQL DML şi DQL ca şi cum ar fi un tabel real. Unii consideră că vizualizările sunt „tabele

virtuale", deoarece se comportă la fel ca tabelele (cu unele restricţii), dar nu există ca tabele fizice.

Sintaxa generală a instrucţiunii CREATE VIEW este:

CREATE [OR REPLACE] VIEW *nume_vizualizare* AS *interogare_sql*

- Cuvântul cheie opţional OR REPLACE elimină necesitatea de a şterge o vizualizare existentă înainte de a o crea din nou. Dacă specificaţi parametrul opţional OR REPLACE şi vizualizarea există deja, este înlocuită, iar dacă nu există, noua vizualizare este adăugată în baza de date.
- Numele vizualizării trebuie să respecte aceleaşi reguli de denumire ca şi tabelele şi alte obiecte ale bazei de date. În interogările SQL sunt specificate numele obiectelor din care sunt selectate datele, dar nu şi tipul acestor obiecte. Ca urmare, numele vizualizărilor trebuie să fie unice pentru toate tabelele şi vizualizările din baza de date. Cu alte cuvinte, numele vizualizărilor şi numele tabelelor trebuie să provină din acelaşi *spaţiu de nume,* adică acelaşi domeniu de nume.
- Interogarea SQL inclusă în definiţia vizualizării poate fi orice instrucţiunea SQL SELECT validă. Vom învăţa despre această instrucţiune SQL esenţială în capitolul 4. Crearea vizualizărilor urmează o cale naturală de evoluţie - lucraţi cu interogarea SQL, faceţi modificările necesare şi rulaţi din nou interogarea până când obţineţi rezultatele dorite. Apoi adăugaţi instrucţiunea CREATE VIEW în faţa interogării cu care aţi lucrat şi rulaţi instrucţiunea pentru a stoca permanent interogarea în baza de date, ca vizualizare. Aceasta este o modalitate foarte productivă (şi plăcută) de a lucra cu bazele de date.

Instrucţiunea ALTER TABLE

După ce creaţi un tabel, aproape orice aţi specificat în instrucţiunea CREATE TABLE poate fi modificat folosind instrucţiunea ALTER TABLE.

Utilizarea instrucţiunii ALTER TABLE este un alt domeniu în care au un rol important stilul şi preferinţele personale. Mulţi administratori de baze de date preferă să folosească instrucţiuni CREATE TABLE cât mai simple, evitând să definească restricţii în instrucţiunile CREATE TABLE. Aceştia adaugă după instrucţiunea CREATE TABLE instrucţiuni ALTER TABLE prin care specifică toate restricţiile necesare (cheie primară, cheie externă, unicitate, verificare şi aşa mai departe). Dezavantajul acestei metode este acela că necesita scrierea unei cantităţi mai mari de cod. Pe de altă parte, instrucţiunea CREATE TABLE este mult mai uşor de înţeles fără restricţii, iar scrierea separată a restricţiilor simplifică refolosirea instrucţiunilor, dacă este nevoie să eliminaţi şi apoi să creaţi din nou restricţiile.

***Adăugarea unei coloane la un tabel**. Definirea coloanei se face cu aceeaşi sintaxă ca şi în cazul instrucţiunii CREATE TABLE.

ALTER TABLE *nume_tabel* ADD (<definiţie_coloană> [,<definiţie_coloană>…]);

Exemplu:

ALTER TABLE CUSTOMER_ACCOUNT ADD (CUSTOMER_HOLD_DATE DATE NULL,

HOLD_PLACED_BY VARCHAR(50));

Modificarea definiţiei unei coloane.

ALTER TABLE *nume_tabel* MODIFY | CHANGE [COLUMN] (<definiţie_coloană> [,<definiţie_coloană>…]);

Exemplu:

ALTER TABLE CUSTOMER_ACCOUNT MODIFY (CUSTOMER_DEPOSIT_AMOUNT NUMERIC(7,2) DEFAULT 0 NOT NULL);

***Adăugarea unei restricţii.** Definiţia restricţiei este identică cu definiţia unei restricţii care ar putea apărea într-o instrucţiune CREATE TABLE.

ALTER TABLE *nume_tabel* ADD CONSTRAINT <definiţie_restricţie>;

Exemplu:

ALTER TABLE CUSTOMER_ACCOUNT ADD CONSTRAINT CK_CUSTOMER

DEPOSIT_AMOUNT CHECK (CUSTOMER_DEPOSIT_AMOUNT >= 0 OR
CUSTOMER_DEPOSIT_AMOUNT IS NULL);

**** Ştergerea cheii primare a unui tabel.*** Dacă cheia primară este referită de restricţii referenţiale, trebuie mai întâi şterse restricţiile respective.

ALTER TABLE *nume_tabel* DROP PRIMARY KEY;

**** Redenumirea unei coloane.***

ALTER TABLE nume_tabel RENAME nume_vechi_coloană TO nume_nou_coloană;

În MySQL este implementată varianta:

ALTER TABLE nume_tabel
CHANGE [COLUMN] nume_*col_vechi* nume_*col_noua column_definition*
[FIRST|AFTER *col_name*]

Instrucţiunea DROP

Instrucţiunea DROP este cea mai simplă dintre instrucţiunile DDL. Sintaxa de bază este:

DROP <tip_obiect> *nume_obiect* [<opţiuni_de_ştergere>]

- Tipul de obiect specifică tipul obiectului care urmează să fie şters, cum ar fi INDEX, TABLE sau VIEW.
- Opţiunile de ştergere sunt specifice fiecărui DBMS. În general, dacă un tabel este referit de o restricţie referenţială, sistemul DBMS nu vă va permite să ştergeţi tabelul

Iată câteva exemple:

DROP TABLE CUSTOMER_ACCOUNT;
DROP TABLE CUSTOMER_ACCOUNT CASCADE CONSTRAINTS; (Oracle)
DROP TABLE CUSTOMER_ACCOUNT CASCADE; (MySQL / PostgreSQL)
DROP INDEX IX_MOVIE_TITLE ON MOVIE;

1.3.2. Instrucţiuni DQL (Data Query Language)

Limbajul SQL de interogare a datelor (DQL - Data Query Language) include o singură comandă, dar una foarte importantă: SELECT. Comanda SELECT este folosită pentru a obţine date din baza de date (fără să le modifice), astfel încât acestea să poate fi prelucrate de o aplicaţie sau afişate pentru un utilizator.

Rezultatul unei instrucţiuni SELECT, numit *set de rezultate,* este returnat întotdeauna sub forma unui tabel (adică rânduri şi coloane). Nu uitaţi că SQL este un limbaj neprocedural, aşa că specificaţi rezultatele pe care vreţi să le obţineţi (adică modul în care vreţi să fie returnat setul de rezultate), nu *şi* modul în care vor fi obţinute acestea.

Instrucţiunea SELECT de bază

Forma elementară a instrucţiunii SELECT conţine două clauze:

- **SELECT [DISTINCT] -** Specifică lista de coloane care urmează să fie returnate în setul de rezultate, separate prin virgule. Puteţi folosi simbolul asterisc (*) în locul listei de coloane pentru a selecta toate coloanele dintr-un tabel sau dintr-o vizualizare. Cuvântul cheie DISTINCT poate fi adăugat după cuvântul cheie SELECT pentru a elimina rândurile duplicate din rezultatele interogării.
- **FROM -** Specifică lista tabelelor sau vizualizărilor din care urmează să fie selectate datele. În locul numelor reale ale tabelelor sau vizualizărilor puteţi folosi *sinonime,* adică pseudonime pentru tabele sau vizualizări definite în baza de date.

Exemplul următor selectează coloanele MOVIE_GENRE_CODE, MPAA_RATING_CODE şi MOVIE_TITLE din tabelul MOVIE.

SELECT MOVIE_GENRE_CODE, MPAA_RATING_CODE, MOVIE_TITLE FROM MOVIE;

Pseudonime pentru numele coloanelor

Se observă din interogarea anterioară că numele coloanelor din tabel apar automat ca titluri ale coloanelor din setul de rezultate. Totuşi, nu este obligatoriu să se întâmple aşa, deoarece

instrucţiunea SQL vă pune la dispoziţie o modalitate simplă de a specifica pseudonime pentru coloane. Pseudonimele *(aliases)* specificate devin numele coloanelor din setul de rezultate. Atenţie la un aspect - pseudonimele nu există decât după rularea instrucţiunii SQL, aşa că nu pot fi folosite în alte părţi ale instrucţiunii SQL. Pseudonimul unei coloane este specificat prin plasarea cuvântului cheie „AS" după numele coloanei în lista SELECT (cu cel puţin un spaţiu înainte si după), urmat de numele pe care vreţi să-l atribuiţi coloanei în setul de rezultate. Iată cum arată instrucţiunea SQL rulată mai devreme, dar cu coloana MOVIE_GENRE_CODE redenumită GENRE şi coloana MPAA_RATING_CODE redenumită RATING.

```
SELECT MOVIE_GENRE_CODE AS GENRE,
MPAA_RATING_CODE AS RATING, MOVIE_TITLE FROM MOVIE;
```

Sortarea rezultatelor

Rezultatele interogărilor sunt deseori mult mai utile dacă specificaţi pentru rândurile returnate o ordine care să aibă o semnificaţie pentru persoana sau aplicaţia care foloseşte informaţiile. Nu există nici o garanţie în privinţa ordinii în care sunt returnate rândurile din setul de rezultate *decât dacă* ordinea dorită este specificată în interogare.

În SQL, acest lucru este făcut prin adăugarea în instrucţiunea SELECT a clauzei **ORDER BY,** cu o listă de una sau mai multe coloane care vor fi folosite pentru sortarea rândurilor în ordine ascendentă sau descendentă, în conformitate cu valorile datelor din coloane.

În cazul instrucţiunii SELECT folosite mai devreme, să presupunem că ar fi utilă ordonarea ascendentă a rândurilor după coloanele MPAA_Rating şi Movie_Genre_Code. Din perspectiva umană, cel mai bine este să plasaţi aceste coloane pe primele poziţii în rezultatele interogării şi să le specificaţi în aceeaşi ordinea în lista ORDER BY (cel puţin în limbile citite de la stânga la dreapta). Astfel, ordonarea rândurilor este evidentă pentru cel care citeşte rezultatele. Mai jos este prezentată instrucţiunea SELECT modificată.

```
SELECT MPAA_RATING_CODE AS RATING, MOVIE_GENRE_CODE AS GENRE,
MOVIE_TITLE FROM MOVIE ORDER BY MPAA_RATING_CODE,
MOVIE_GENRE_CODE;
```

Utilizarea clauzei WHERE pentru filtrarea rezultatelor

SQL foloseşte clauza "WHERE pentru a filtra rândurile ce urmează să fie afişate. Aşa cum aţi văzut deja, o interogare fără o clauză WHERE returnează un set de rezultate care conţine toate rândurile din tabelele sau vizualizările referite în clauza FROM. Dacă este inclusă o clauză WHERE, sunt folosite regulile algebrei booleene, numite astfel după numele logicianului George Boole, evaluând clauza WHERE pentru fiecare rând de date. În rezultatele interogării sunt afişate numai rândurile pentru care clauza WHERE este evaluată la valoarea logică „adevărat".

Operatori de comparare

Operatorii de comparare sunt folosiţi în clauza WHERE pentru compararea a două valori, având ca rezultat o valoare logică de „adevărat" sau „fals". Cele două valori comparate pot fi constante furnizate în clauza WHERE, valori ale unor coloane din baza de date sau combinaţii ale celor două. Operatorii de comparare care pot fi folosiţi în clauza WHERE sunt prezentaţi în tabelul următor:

O	Descriere
=	Egal cu
<	Mai mic decât
<=	Mai mic sau egal
>	Mai mare decât
>=	Mai mare sau egal
!=	Diferit de
<>	Diferit de (standard

Exemple:

- Afişaţi toate filmele pentru care MPAA_Rating are valoarea PG-13

```
SELECT MPAA_RATING_CODE AS RATING, MOVIE_TITLE
FROM MOVIE WHERE MPAA_RATING_CODE = 'PG-13'
ORDER BY MOVIE_TITLE;
```

- Afişaţi toate filmele cu preţul de vânzare cu amănuntul pentru formatul DVD (DVD Retail Price) mai mic de 19.99, în ordinea descrescătoare a preţurilor.
  SELECT RETAIL_PRICE_DVD, MOVIE_TITLE
  FROM MOVIE WHERE RETAIL_PRICE_DVD < 19.99
  ORDER BY RETAIL_PRICE_DVD DESC;

Operatori conjunctivi

Uneori sunt necesare condiţii multiple pentru a îngusta setul de rezultate al unei interogări. Atunci când sunt folosite mai multe condiţii, ele trebuie să fie combinate din punct de vedere logic în clauza WHERE, iar aceasta este sarcina *operatorilor conjunctivi.* Aceşti operatori sunt:

- **AND (ŞI)** Clauza WHERE este evaluată ca „adevărată" dacă *toate* condiţiile conectate cu operatorul AND sunt adevărate.
- **OR (SAU)** Clauza WHERE este evaluată ca „adevărată" dacă *oricare* din condiţiile conectate cu operatorul OR este adevărată.

Lucrurile devin complicate dacă operatorii AND şi OR sunt combinaţi în aceeaşi clauză WHERE. Operatorul AND are prioritate mai mare şi, ca urmare, este evaluat înaintea operatorilor OR. Condiţiile din interiorul parantezelor sunt evaluate întotdeauna primele.

Iată un ***exemplu*** de folosire a operatorilor conjunctivi;

- Afişaţi toate filmele pentru care categoria MPAA este PG-13 *şi* preţul de vânzare cu amănuntul pentru formatul DVD este 19.99 sau mai mic, în ordinea crescătoare a preţurilor.
  SELECT MPAA_RATING_CODE AS RATING, RETAIL_PRICE_DVD AS PRICE, MOVIE_TITLE FROM MOVIE WHERE MPAA_RATING_CODE = 'PG-13'
  AND RETAIL_PRICE_DVD <= 19.99 ORDER BY RETAIL_PRICE_DVD;

Operatori logici

Operatorii logici folosesc cuvinte cheie în locul simbolurilor la formarea expresiilor de comparare. Operatorii logici disponibili în majoritatea implementărilor SQL sunt prezentaţi în secţiunile care urmează. La oricare dintre aceşti operatori poate fi adăugat cuvântul cheie NOT, pentru a inversa valoarea logică a comparaţiei.

IS NULL

Operatorul IS NULL este folosit pentru a determina dacă o valoare este nulă. Este important să reţineţi că valorile nule din baza de date nu sunt egale cu nimic altceva, nici chiar cu alte valori nule. Din această cauză, o condiţie precum "= NULL" este întotdeauna incorectă. Niciodată nimic nu este egal cu o valoare nulă.

Exemplu:

- Găsiţi toate conturile de clienţi active, adică toate conturile pentru care coloana DATE_TERMINATED conţine o valoare nulă:
  SELECT CUSTOMER_ACCOUNT_ID FROM CUSTOMER_ACCOUNT
  WHERE DATE_TERMINATED IS NULL;
- Găsiţi toate conturile de clienţi inactive, adică toate conturile pentru care coloana DATE_TERMINATED conţine o altă valoare decât NULL:
  SELECT CUSTOMER_ACCOUNT_ID FROM CUSTOMER_ACCOUNT
  WHERE DATE_TERMINATED IS NOT NULL;

BETWEEN

Operatorul BETWEEN este folosit pentru a determina dacă o valoare se încadrează într-un interval specificat. Intervalul este specificat folosind o valoare minimă şi o valoare maximă, fiind un interval *inclusiv,* ceea ce înseamnă că include şi valorile specificate. Aceasta este o prescurtare elegantă, uşor de citit şi de înţeles, pentru specificare unei condiţii de interval. De exemplu, condiţia „WHERE MOVIE_ID BETWEEN 7 AND 9" este aceeaşi cu condiţia „WHERE MOVIE_ID >= 7 AND MOVIE_ID <= 9.

Exemplu:

- Afişaţi toate filmele cu preţul de vânzare cu amănuntul pentru formatul DVD (coloana RETAIL_PRICE_DVD) între 14.99 şi 19.99, ordonate crescător după preţ. Observaţi că în setul de rezultate sunt incluse si filmele pentru care preţul

este exact 14.99 sau 19.99.

```
SELECT MOVIE_TITLE, RETAIL_PRICE_DVD
FROM MOVIE WHERE RETAIL_PRICE_DVD BETWEEN 14.99 AND 19.99
ORDER BY RETAIL_PRICE_DVD;
```

LIKE

Operatorul LIKE este folosit pentru a compara o valoare de tip caracter cu un tipar, returnând valoarea logică „adevărat" dacă valoarea de tip caracter se încadrează în tipar şi „fals" în caz contrar. Pentru definirea tiparului pot fi folosite două caractere de înlocuire:

- Liniuţa de subliniere (_). Caracterul liniuţa de subliniere poate fi folosit drept caracter de înlocuire poziţional, ceea ce înseamnă că se potriveşte cu orice caracter aflat pe poziţia respectivă în şirul de caractere evaluat.
- Procent *(%)*. Simbolul procent (%) poate fi folosit drept caracter de înlocuire nepoziţional, ceea ce înseamnă că se potriveşte cu orice număr de caractere, indiferent de lungime.

Exemplu de utilizare a operatorului LIKE:

```
SELECT 'David!' LIKE 'David_';      // returnează valoarea 1
SELECT 'David!' LIKE 'David\_';          // returnează valoarea 0
```

- Afişaţi toate titlurile de filme care conţin şirul de caractere „on":

```
SELECT MOVIE_TITLE FROM MOVIE WHERE MOVIE_TITLE LIKE '%on%' ;
```

sau

```
SELECT 'David!' LIKE '%D%v%';        // returnează valoarea 1
```

IN

Operatorul IN este folosit pentru a determina dacă o valoare face parte dintr-o listă de valori. Lista poate fi specificată ca valori literale, folosind o listă de valori separate prin virgule şi încadrată între paranteze, sau poate fi selectată din baza de date folosind o *subselecţie* (o *subinterogare),* care este o interogare în cadrul unei alte interogări.

Exemplu:

- Afişaţi toate filmele pentru care MOVIE_GENRE_CODE este Drama, Forgn sau Rmce. Evident, această interogare ar putea fi scrisă folosind trei condiţii de egalitate, separate prin operatorul logic OR, dar este mult mai simplu să o scriem cu ajutorul operatorului IN:

```
SELECT MOVIE_GENRE_CODE AS GENRE, MOVIE_TITLE FROM MOVIE
WHERE MOVIE_GENRE_CODE IN ('Drama','Forgn',' Rmce')
ORDER BY MOVIE_GENRE_CODE, MOVIE_TITLE;
```

EXISTS

Operatorul EXISTS este folosit pentru a determina dacă o subinterogare conţine înregistrări. Dacă în setul de rezultate al subinterogării nu există, operatorul returnează valoarea logică „false"; dacă setul de rezultate conţine cel puţin un rând, valoarea logică devine „adevărat".

Exemplu:

Proprietarul magazinului a auzit că filmul *The Last Samurai* se închiriază bine atât în format DVD, cât şi-n format VHS şi vrea să se asigure că în inventarul magazinului există o copie VHS. Tabelul MOVIE_COPY conţine un rând pentru fiecare copie a unui film din inventarul magazinului, aşa că puteţi folosi o subinterogare pentru a afla dacă există copii VHS ale filmului în inventar.

De cele mai multe ori, operatorul EXISTS este folosit în conjuncţie cu o formă mai complexă de subinterogare, numită *subinterogare corelată (correlated subquery),* în care valorile datelor din interogarea externă (MOVIE_ID, în acest caz) sunt comparate cu rândurile din interogarea internă. Este suficient să ştiţi că subinterogarea va returna un rând dacă există o copie VHS în stoc şi nu va returna nici un rând dacă nu există o astfel de copie. Aceeaşi interogare ar putea fi scrisă şi folosind operatorul IN sau folosind o *uniune,* care leagă rândurile din două tabele.

```
SELECT m.MOVIE_ID, m.MOVIE_TITLE FROM MOVIE m
```

WHERE m.MOVIE_TITLE = 'The Last Samurai' AND EXISTS
(SELECT c.MOVIE_ID FROM MOVIE_COPY c WHERE m.MOVIE_ID = c.MOVIE_ID)

Operatori aritmetici

În SQL, operatorii aritmetici sunt folosiţi pentru efectuarea calculelor matematice - la fel ca şi-n formulele dintr-o foaie de calcul tabelar sau într-un limbaj de programare, precum Java sau C. Cei patru operatori aritmetici din SQL sunt:

Operator	Descriere
+	adunare
-	scădere
*	înmulţire
/	împărţire

Funcţii SQL elementare

O funcţie este un tip special de program, care returnează o singură valoare de fiecare dată când este apelată. Termenul provine de la conceptul matematic al unei funcţii. În SQL, funcţiile necesită întotdeauna specificarea unei expresii, care deseori include numele unei coloane. Cel mai des, funcţiile sunt folosite în lista de coloane a unei instrucţiuni SELECT. Sunt apelate pentru fiecare rând prelucrat de interogare şi, ca urmare, returnează o singură valoare pentru fiecare rând din setul de rezultate.

Funcţii pentru caractere

Funcţiile pentru caractere sunt numite astfel deoarece manipulează date de tip text.

Concatenarea şirurilor de caractere

Funcţia de *concatenare a şirurilor de caractere* reuneşte mai multe şiruri de caractere pentru a forma o singură valoare în rezultatele interogării. Funcţia standard de concatenare a şirurilor de caractere din SQL este apelată cu două bare verticale (||), dar există şi excepţii, cum ar fi Microsoft SQL Server, care foloseşte semnul plus (+) pentru concatenarea şirurilor de caractere.

UPPER

Funcţia UPPER transformă literele dintr-un şir de caractere în litere mari. Numerele şi caracterele speciale sunt lăsate ca atare.

Exemplu:

- Afişaţi comediile (MOVIE_GENRE_CODE = 'Comdy') scriind titlurile cu majuscule. Remarcaţi că am folosit un pseudonim pentru a ne asigura că numele coloanei MOVIE_TITLE apare în setul de rezultate.
  SELECT UPPER(MOVIE_TITLE) AS MOVIE_TITLE FROM MOVIE
  WHERE MOVIE_GENRE_CODE ='Comdy';

Observaţie: La folosirea funcţiilor SQL în condiţiile WHERE, în cele mai multe situaţii, pentru o coloană căreia îi este aplicată o funcţie nu poate fi folosită indexarea. Ca urmare, în cazul tabelelor mari, utilizarea funcţiilor în condiţiile WHERE poate duce la probleme de performanţă cu adevărat memorabile. În concluzie, trebuie să specificaţi întotdeauna clauza ORDER BY dacă ordinea în care apar rezultatele este importantă.

LOWER

Funcţia LOWER este inversa funcţiei UPPER - transformă literele dintr-un şir de caractere în litere mici.

Exemplu de utilizare a funcţiei LOWER:

- Afişaţi comediile (MOVIE_GENRE_CODE = 'Comdy) scriind titlurile cu minuscule. Remarcaţi că am folosit un pseudonim pentru a ne asigura că numele coloanei MOVIE_TITLE apare în setul de rezultate.
  SELECT LOWER(MOVIE_TITLE) AS MOVIE_TITLE FROM MOVIE
  WHERE MOVIE_GENRE_CODE ='Comdy';

SUBSTR

Funcţia SUBSTR apare în majoritatea implementărilor SQL, dar uneori are un nume puţin diferit. De exemplu, funcţia se numeşte SUBSTRING în Microsoft SQL Server, Sybase Adaptive Server şi MySQL, dar SUBSTR în Oracle şi DB2, Funcţia returnează o porţiune a şirului de caractere, în funcţie de parametrii furnizaţi, care specifică numele coloanei, poziţia de început a subşirului în datele coloanei şi lungimea subşirului returnat (numărul de caractere). Deşi este o utilizare mai puţin obişnuită, funcţia SUBSTR acceptă şi un şir de caractere literal în locul numelui unei coloane. Iată forma generala a funcţiei, urmată de un exemplu:

SUBSTR (*numele_coloanei, poziţia_ de_ început, lungimea_subşirului)*

- în tabelul PERSON, unele persoane au al doilea nume în întregime, alţii au numai iniţiala. Afişaţi numele complet al persoanelor al căror nume de familie începe cu litera „B", sub forma unui singur şir de caractere care conţine prenumele, iniţiala şi numele de familie.
  SELECT PERSON_GIVEN_NAME as SUBSTRING(PERSON_MIDDLE_NAME,1,1)
  AS INITIALA,' ', PERSON_FAMILY_NAME AS NUME
  FROM PERSON WHERE SUBSTRING(PERSON_FAMILY_NAME,1,1)='B'

LENGTH

Funcţia LENGTH returnează lungimea unui şir de caractere. Reţineţi că implementările Microsoft SQL Server şi Sybase Adaptive Server folosesc numele LEN pentru versiunea proprie a acestei funcţii.

Exemple:

- Afişaţi lungimea titlului pentru filmul a cărui valoare MOVIE_ID este 1. Presupunem că folosiţi o bază de date Oracle, DB2 sau MySQL.
  SELECT MOVIE_TITLE, LENGTH(MOVIE_TITLE) AS LENGTH FROM MOVIE WHERE MOVIE_ID = 1;
- Afişaţi lungimea titlului pentru filmul a cărui valoare MOVIE_ID este 1. Presupunem că folosiţi o baza de date Microsoft SQL Server.
  SELECT MOVIE_TITLE, LEN(MOVIE_TITLE) AS LENGTH FROM MOVIE WHERE LEN(MOVIE_TITLE)<10;

Funcţii matematice

Funcţiile matematice manipulează valori numerice, în conformitate cu regulile matematicii.

Funcţia **ROUND** este prezentată în detaliu, inclusiv cu un exemplu, apoi urmează un tabel cu funcţiile matematice existente în majoritatea implementărilor SQL.

ROUND

Funcţia ROUND rotunjeşte o valoare la un număr specificat de zecimale. Valoarea numerică este furnizată prin primul parametru, iar numărul de zecimale prin cei de-al doilea. În continuare este prezentat formatul general al funcţiei ROUND, urmat de un exemplu.

ROUND (*expresie_numerică, nr_de_poziţii_zecimale)*

- Care este costul mediu al unei copii a filmului *The Last Samurai,* rotunjit la două zecimale?
  SELECT ROUND((RETAIL_PRICE_VHS + RETAIL_PRICE_DVD) / 2, 2) AS AVG_COST FROM MOVIE WHERE MOVIE_TITLE = 'The Last Samurai';

Tabelul care urmează prezintă funcţiile matematice cel mai des întâlnite. Pentru toate, sintaxa generală este aceeaşi:

NUME_FUNCŢIE *(expresie)*

Funcţie	**Descriere**
ABS	Valoarea absolută a unui număr dat
COS	Cosinusul trigonometric al unui unghi specificat în radiani
EXP	Valoarea exponenţială a unui număr dat
POWER	Ridică un număr la o putere (numărul şi puterea sunt furnizate ca parametri)
SIN	Sinusul trigonometric al unui unghi specificat în radiani
TAN	Tangenta trigonometrică a unui unghi specificat în *radiani*

Funcţii de conversie

Funcţiile de conversie transformă date dintr-un tip de date în altul.

CAST

Funcţia CAST transformă date dintr-un tip de date în altul. Reţineţi că această funcţie nu este implementată de DB2.

CAST (*expresie* AS *tip de date)*

- Afişaţi preţul pentru formatul DVD al filmului *The Last Samurai,* cu un simbol dolar în faţa sumei. Valoarea numerică trebuie să fie convertită într-un şir de caractere pentru a putea fi concatenată cu o valoare literală conţinând simbolul dolar.

```
SELECT CONCAT('$: ', CAST(RETAIL_PRICE_DVD AS char(6))) as Price
FROM MOVIE WHERE MOVIE_TITLE = 'The Last Samurai'
sau
SELECT NOW();
SELECT concat('Data: ', CAST(NOW() AS DATE));
```

Funcţii de agregare şi gruparea rândurilor

O funcţie de agregare este o funcţie care combină mai multe rânduri de date într-un singur rând. Tabelul următor prezintă funcţiile de agregare acceptate în majoritatea implementărilor SQL:

Funcţie	**Descriere**
AVG	Calculează valoarea medie pentru o coloană sau o expresie.
COUNT	Numără valorile dintr-o coloană. Puteţi folosi cuvântul cheie distinct pentru a număra valorile unice, în locul tuturor valorilor (rândurilor) dintr-o coloană.
MAX	Găseşte valoarea maximă dintr-o coloană.
MIN	Găseşte valoarea minimă dintr-o coloană.
SUM	Însumează valorile dintr-o coloană.

Exemple:

- Care este preţul mediu al unui DVD? Observaţi că funcţiile ROUND şi AVG sunt imbricate, astfel încât să obţineţi rezultatul în dolari şi cenţi.

```
SELECT ROUND(AVG(RETAIL_PRICE_DVD),2)  AS AVG_PRICE FROM MOVIE;
```

- Câte filme există în tabelul MOVIE?

```
SELECT COUNT(*)  AS NUM_MOVIES FROM MOVIE;
```

- Câte genuri diferite de filme sunt reprezentate în tabelul MOVIE? Observaţi folosirea cuvântului cheie DISTINCT, astfel încât sistemul DBMS să numere numai valorile MOVIE_GENRE_CODE unice.

```
SELECT COUNT(DISTINCT(MOVIE_GENRE_CODE)) AS NUM_GENRES FROM
MOVIE;
```

Clauza GROUP BY

Aşa cum aţi observat, dacă folosiţi o funcţie de agregare, ca atare, într-o interogare, obţineţi ca rezultat un singur rând din întreaga interogare. Este logic, deoarece sistemul RDBMS nu ştie ce alte rezultate aţi dori să obţineţi, decât dacă-i spuneţi acest lucru - şi tocmai acesta este scopul clauzei GROUP BY. Aceasta cere sistemului DBMS *să grupeze* rândurile selectate de interogare pe baza valorilor din una sau mai multe coloane şi să aplice funcţia (sau funcţiile) de agregare fiecărui grup, returnând un rând pentru fiecare grup din setul de rezultate. Este ca şi cum aţi cere subtotaluri pentru fiecare departament, în locul unui singur total pentru întreaga companie, dar, aşa cum aţi văzut, funcţiile de agregare pot face multe alte lucruri, nu numai însumarea unor valori. Sistemul DBMS va ordona rândurile selectate de interogare după coloanele din clauza GROUP BY, aşa că grupurile vor fi returnate în ordine ascendentă, exceptând cazul în care adăugaţi o clauză ORDER BY care specifică un alt mod de ordonare.

Exemplu:

- Afişaţi fiecare cod de gen, împreună cu numărul de filme asociate fiecărui cod.
  SELECT MOVIE_GENRE_CODE AS GENRE, COUNT(*) AS COUNT FROM MOVIE GROUP BY MOVIE_GENRE_CODE;

Operatori pentru interogări compuse

Uneori este util să rulăm interogări multiple şi să combinăm rezultatele într-un singur set de rezultate.

UNION

Operatorul UNION adaugă rândurile din setul de înregistrări al unei interogări la cel al unei alte înregistrări şi, în acelaşi timp, elimină rândurile duplicate, într-un mod similar cu cel al cuvântului cheie DISTINCT. Operaţia este permisă numai dacă interogările sunt compatibile din punctul de vedere al uniunii, ceea ce înseamnă că au acelaşi număr de coloane şi că tipurile de date ale coloanelor corespondente sunt compatibile.

Exemplu:

- Afişaţi pe o singură coloană toate valorile nenule pentru taxa de închiriere şi taxa de întârziere din tabelul MOVIE_RENTAL
  SELECT RENTAL_FEE AS FEE FROM MOVIE_RENTAL
  WHERE RENTAL_FEE IS NOT NULL UNION
  SELECT LATE_OR_LOSS_FEE AS FEE1 FROM MOVIE_RENTAL
  WHERE LATE_OR_LOSS_FEE IS NOT NULL;

UNION ALL

UNION ALL funcţionează la fel ca şi operatorul UNION, exceptând faptul că rândurile duplicate *nu sunt* eliminate.

INTERSECT

Operatorul INTERSECT găseşte valorile selectate dintr-o interogare, care apar şi într-o altă interogare. În esenţă, găseşte intersecţia valorilor din cele două interogări. Totuşi, doar un număr mic de sisteme DBMS (cele mai importante fiind Oracle şi DB2) implementează acest operator. Nu este implementat în Microsoft SQL Server sau MySQL.

EXCEPT

EXCEPT este operatorul standard ANSI/ISO care găseşte diferenţele dintre două seturi de rezultate, returnând, în esenţă, valorile din prima interogare care nu apar în cea de-a doua interogare. Foarte puţine sisteme DBMS implementează acest operator. În unele implementări, precum Oracle, operatorul se numeşte MINUS, nu EXCEPT.

1.3.3. Interogări din tabele multiple

Uniuni (join)

O *uniune (join)* este o operaţie într-o bază de date relaţională care combină coloane din două sau mai multe tabele în rezultatele unei singure interogări. O uniune apare de fiecare dată când clauza FROM a unei instrucţiuni SELECT specifică numele mai multor tabele.

Uniuni de egalitate (equijoin)

Avem o *uniune de egalitate (equijoin),* numită şi *uniune internă (innerjoin),* atunci când legăm una sau mai multe coloane dintr-un tabel (de obicei, o cheie externă) cu coloane similare dintr-un alt tabel (de obicei, cheia primară), folosind condiţia de *egalitate* (cu alte cuvinte, considerăm că acele coloane se potrivesc dacă valorile datelor sunt egale).

Aceasta este cea mai des folosită formă de uniune. Totuşi, termenul *uniune de egalitate (equijoin)* este rareori folosit în afara mediilor academice, fiind preferate denumirile *uniune internă (innerjoin)* sau *uniune standard (standard join).*

Există două modalităţi de specificare a coloanelor corespondente: folosind clauza WHERE sau folosind clauza JOIN. Clauza JOIN a fost adăugată relativ recent în standardul SQL, aşa că programatorii mai vechi sunt obişnuiţi cu metoda bazată pe clauza WHERE.

Realizarea uniunilor folosind clauza WHERE

Folosirea clauzei WHERE pentru unirea tabelelor seamănă cu folosirea acesteia pentru eliminarea rândurilor de care nu aveţi nevoie din setul de rezultate. Totuşi, există unele diferenţe. În primul rând, în condiţia WHERE comparaţi o coloană cu o altă coloană, nu o coloană cu o constantă sau o expresie. În al doilea rând, atunci când coloanele din cele două tabele au acelaşi nume (o soluţie recomandată) trebuie să specificaţi numele complet al coloanelor (adică numele cu calificator), astfel încât motorul SQL să ştie care dintre cele două coloane este referită. Cea mai simplă formă de calificator este chiar numele tabelului, separat cu un caracter punct de numele coloanei.

În continuare este prezentat un ***exemplu*** de uniune realizată prin clauza WHERE, cu numele coloanelor specificate complet, folosind numele tabelelor. Observaţi că interogarea selectează coloanele MOVIE_ID şi MOVIE_TITLE din tabelul MOVIE şi genul corespunzător (MOVIE_GENRE_DESCRIPTION) din tabelul MOVIE_GENRE.

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
FROM MOVIE, MOVIE_GENRE
WHERE  MOVIE.MOVIE_GENRE_CODE=MOVIE_GENRE.MOVIE_GENRE_CODE
ORDER BY MOVIE_ID;
```

Observaţie: Folosirea numelor complete ale tabelelor pentru specificarea coloanelor poate fi obositoare şi consumatoare de timp, mai ales deoarece numele tabelelor pot avea 30 sau mai multe caractere în sistemele DBMS moderne. Din aceasta cauză, în SQL este permisă folosirea pseudonimelor *(aliases)* pentru numele tabelelor.

Exemplul următor prezintă instrucţiunea anterioară, după adăugarea pseudonimelor pentru numele tabelelor. Deşi nu era necesar, pseudonimele au fost adăugate şi în lista de coloane din clauzele SELECT şi ORDER BY, pentru a vă arăta cum sunt folosite.

```
SELECT A.MOVIE_ID, B.MOVIE_GENRE_DESCRIPTION AS GENRE, A.MOVIE_TITLE
FROM MOVIE A, MOVIE_GENRE B
WHERE A.MOVIE_GENRE_CODE=B.MOVIE_GENRE_CODE
ORDER BY A.MOVIE_ID;
```

Realizarea uniunilor folosind clauza JOIN

Clauza JOIN este scrisă ca o referinţă de tabel în clauza FROM şi, în esenţă, combină lista de tabele din clauza FROM şi condiţia de legătură scrisă anterior în clauza WHERE într-o singură clauză.

Sintaxa generală a clauzei JOIN pentru o uniune internă:

nume_tabel [INNER] JOIN nume_tabel { ON condiţie | USING (nume_coloană [,nume_coloană…]) }

Observaţi cele două opţiuni. Clauza ON permite specificarea unei condiţii similare cu cea din clauza WHERE folosită în exemplul anterior. Clauza USING, pe de altă parte, specifică numele coloanelor folosite pentru legarea rândurilor. Totuşi, clauza USING funcţionează numai atunci când coloanele pe care se face legătura au nume identice în ambele tabele. Iată câteva ***exemple:***

- ***JOIN cu condiţie ON:***
  ```
  SELECT MOVIE_ID,  MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
  FROM MOVIE JOIN MOVIE_GENRE ON
  MOVIE.MOVIE_GENRE_CODE = MOVIE_GENRE.MOVIE_GENRE_CODE
  ORDER BY MOVIE_ID;
  ```
- ***JOIN folosind cuvântul cheie USING*** (în locul condiţiei ON) - o scurtătură elegantă atunci când coloanele din cele două tabele au acelaşi nume.
  ```
  SELECT MOVIE_ID,  MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
  FROM MOVIE JOIN MOVIE_GENRE USING (MOVIE_GENRE_CODE)
  ORDER BY MOVIE_ID;
  ```
- ***JOIN cu cheie externă pe mai multe coloane.*** Aceasta interogare afişează lista cu copiile filmelor din tabelul MOVIE_COPY cave nu au fost vândute (coloana DATE_SOLD conţine o

valoare nulă) şi care au fost închiriate (uniune cu tabelul MOVIE_RENTAL) dar nu au fost încă returnate (coloana RETURNED_DATE conţine o valoare nulă).

```
SELECT MOVIE_ID,  COPY_ NUMBER, DUE_DATE
FROM MOVIE_COPY JOIN MOVIE_RENTAL USING (MOVIE_ID, COPY_NUMBER)
WHERE DATE_SOLD IS NULL  AND  RETURNED_DATE  IS NULL;
```

Uniuni naturale

O *uniune naturală (natural join)* se bazează pe toate coloanele cu acelaşi nume din două tabele. În esenţă, toate uniunile de egalitate pe care le-aţi văzut deja sunt uniuni naturale. Totuşi, sintaxa unei uniuni naturale este mult mai simplă, deoarece nu este necesar să specificaţi o condiţie sau o listă de coloane - se înţelege de la sine care sunt coloanele folosite.

Exemplu: uniunea tabelelor MOVIE şi MOVIE_GENRE, rescrisă sub forma unei uniuni naturale:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
FROM MOVIE NATURAL JOIN MOVIE_GENRE
ORDER BY MOVIE_ID;
```

Uniuni externe (outer joins)

O *uniune externă (outer join)* - pentru care un nume mai potrivit ar fi uniune *inclusivă* - include în setul de rezultate şi rândurile pentru care nu există legături din cel puţin unul dintre tabele. Atunci când există rânduri fără legături, datele selectate din tabelul în care nu a fost găsită o legătură primesc valoarea nulă. Există trei tipuri de bază:

- ***Uniune externă către stânga (left outer join).*** Returnează toate rândurile din tabelul din stânga (cel specificat primul în clauza JOIN), împreună cu toate rândurile din tabelul din dreapta pentru care poate fi găsită o legătură.
- ***Uniune externă către dreapta (right outer join).*** Returnează toate rândurile din tabelul din dreapta (cel specificat al doilea în clauza JOIN), împreună cu toate rândurile din tabelul din stânga pentru care poate fi găsită o legătură. În esenţă, o uniune externă către stânga poate fi rescrisă ca o uniune externă către dreapta inversând ordinea de specificare a tabelelor şi înlocuind cuvântul cheie LEFT cu RIGHT.
- ***Uniune externă completă (full outer join).*** Returnează toate rândurile din ambele tabele. Acest tip de uniune este cel mai puţin probabil să fie acceptat de implementarea SQL pe care o aveţi, deoarece sintaxa ei standard este mai nouă decât a celorlalte două. Este esenţial să înţelegeţi că această uniune nu este acelaşi lucru cu un produs cartezian, care leagă *fiecare* rând dintr-un tabel cu *fiecare* rând din celălalt tabel. O uniune externă completă (full outer join) leagă fiecare rând dintr-un tabel cu zero sau mai multe rânduri *corespondente* din celălalt tabel. În realitate, nu veţi întâlni prea multe situaţii în care să folosiţi o uniune externă completă, dar aceasta poate fi utilă dacă între două tabele aveţi o relaţie opţională în ambele direcţii.

  Sintaxa generală pentru o uniune externă este:

  ```
  nume_tabel {RIGHT | LEFT | FULL} [OUTER] JOIN nume_tabel
  {ON conditie | USING (nume_coloana [, nume_coloana …] ) }
  ```

  Exemplu de uniuni externe:

- Afişaţi toate descrierile genurilor de filme, împreună cu filmele asociate fiecărui gen. Observaţi în setul de rezultate rândurile care nu au nici o valoare în coloana MOVIE_TITLE - acestea sunt genurile de filme care nu au filme asociate. Dacă implementarea DBMS nu acceptă uniuni realizate cu cuvântul cheie USING, modificaţi instrucţiunea astfel încât să utilizeze cuvântul cheie ON.

  ```
  SELECT MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE FROM
  MOVIE_GENRE LEFT OUTER JOIN MOVIE USING (MOVIE_GENRE_CODE);
  ```

  Auto-uniuni (self joins)

  O *auto-uniune (self join)* este o uniune a unui tabel cu el însuşi. Acest tip de uniune poate

părea ciudat la prima vedere, dar uneori există relaţii în care cheia primară şi cheia externă se află în acelaşi tabel. Acestea se numesc *relaţii recursive* şi există o asemenea relaţie şi în baza de date a magazinului de produse video.

Exemplu:

Tabelul EMPLOYEE are o coloană numită SUPERVISOR_PERSON_ID, care este o cheie externă către coloana PERSON_ID din acelaşi tabel. Este folosită pentru a lega fiecare angajat de şeful direct, care, desigur, este un alt angajat, ceea ce înseamnă că şi şeful respectiv are o coloană în tabelul EMPLOYEE. Interogarea următoare afişează trei coloane din tabelul EMPLOYEE, inclusiv PERSON_ID şi SUPERVISOR_PERSON_ID:

```
SELECT PERSON_ID, EMPLOYEE_HOURLY_RATE AS HOURLY_RATE,
SUPERVISOR_PERSON_ID FROM EMPLOYEE;
```

Acum să presupunem că magazinul trebuie să producă un raport cu diferenţele de salariu dintre şefi şi subordonaţi. Puteţi lega înregistrarea fiecărui angajat de cea a şefului direct, astfel încât să obţineţi plata pe oră a şefului. Iată interogarea care face acest lucru:

```
SELECT A.PERSON_ID, A.EMPLOYEE_HOURLY_RATE AS HOURLY_RATE,
        B.EMPLOYEE_HOURLY_RATE AS SUPV_HOURLY_RATE
        FROM EMPLOYEE A JOIN EMPLOYEE B
        ON A.SUPERVISOR_PERSON_ID = B.PERSON_ID;
```

Iată şi interogarea finală, care include calcularea diferenţei de salariu şi realizează o uniune cu tabelul PERSON pentru a obţine numele angajaţilor;

```
SELECT A.PERSON_ID,  C.PERSQN_GIVEN_NAME AS  FIRST_NAME,
        C.PERSON_FAMILY_NAME AS  LAST_NAME,
        B.EMPLOYEE_HOURLY_RATE  -  A.EMPLOYEE_HOURLY_RATE AS
RATE_DIFF
        FROM EMPLOYEE A JOIN EMPLOYEE  B
        ON A.SUPERVISOR_PERSON_ID =  B.PERSON_ID
        JOIN  PERSON  C ON A.PERSON_ID  =  C.PERSON_ID;
```

Uniuni încrucişate (cross joins)

O *uniune încrucişată (cross join)* nu este altceva decât sintaxa standard pentru un produs cartezian. Interogarea care unea tabelele MOVIE şi MOVIE_GENRE şi obţinea un produs cartezian poate fi rescrisă ca mai jos, folosind o uniune încrucişată:

```
SELECT MOVIE_ID, MOVIE_GENRE_DESCRIPTION AS GENRE, MOVIE_TITLE
        FROM MOVIE CROSS JOIN MOVIE_GENRE ORDER BY MOVIE_ID;
```

Subinterogări

O caracteristică foarte puternică a limbajului SQL sunt *subinterogările* (numite şi *subselecţii),* care, aşa cum sugerează şi numele, se referă la o instrucţiune SELECT care conţine o instrucţiune SELECT subordonată. De obicei, subinterogările sunt folosite în clauza WHERE, ca modalitate de limitare a rândurilor returnate în setul de rezultate al interogării externe. Aceasta poate fi o modalitate foarte flexibilă de selectare a datelor.

O regulă esenţială de sintaxă este ca subinterogarea să fie încadrată de paranteze. O altă idee importantă pe care trebuie să o înţelegeţi este că orice operaţie efectuată cu o subinterogare poate fi făcută şi printr-o uniune.

Subinterogări necorelate

O *subinterogare necorelată* este o subinterogare în care interogarea internă nu face nici o referire la interogarea externă care o conţine. Aceasta înseamnă că poate fi rulată mai întâi interogarea internă, apoi setul de rezultate obţinut poate fi folosit în interogarea externă.

Exemplu:

- Afişaţi toate limbile în care nu există nici un film în inventarul magazinului de produse video.

```
SELECT LANGUAGE_CODE, LANGUAGE_NAME
FROM LANGUAGE WHERE LANGUAGE_CODE NOT IN
(SELECT DISTINCT LANGUAGE_CODE FROM MOVIE_LANGUAGE)
ORDER BY LANGUAGE_CODE;
```

Subinterogări corelate

O *subinterogare corelată* este o subinterogare în care interogarea internă referă valorile furnizate de interogarea externă. Aceste subinterogări sunt mult mai puţin eficiente decât subinterogările necorelate, deoarece interogarea internă trebuie să fie apelată pentru fiecare rând găsit de interogarea externă.

Exemplu:

- Proprietarul magazinului vrea să transmită prin poştă un cupon valoric tuturor clienţilor care au plătit mai mult de 15$ pentru o singură tranzacţie de închiriere.

```
SELECT DISTINCT CUSTOMER_ACCOUNT_ID
FROM CUSTOMER_TRANSACTION A
WHERE 15 < (SELECT SUM(RENTAL_FEE) FROM MOVIE_RENTAL B
WHERE A.TRANSACTION_ID = B.TRANSACTION_ID);
```

Vizualizări în linie

Foarte puţine implementări, printre care Oracle, permit folosirea unei subinterogări în clauza FROM a unei interogări, într-o construcţie numită ***vizualizare în linie,*** în esenţă, această construcţie permite ca setul de rezultate al unei interogări să fie tratat ca şi cum ar fi un tabel sau o vizualizare predefinită.

Exemplu:

- Această interogare află numărul maxim de închirieri ale unui singur film:

```
SELECT MAX(RENTAL_COUNT) AS MAX_RENTAL_COUNT FROM
(SELECT MOVIE_ID, COUNT(*) AS RENTAL_COUNT
FROM MOVIE_RENTAL GROUP BY MOVIE_ID);
```

1.3.4. Funcţii SQL avansate

Funcţii pentru caractere

Funcţiile pentru caractere operează asupra datelor de tip text.

REPLACE

Funcţia REPLACE caută un şir de caractere şi înlocuieşte caracterele găsite cu caracterele din şirul de înlocuire. Iată sintaxa generală a funcţiei;

REPLACE *(şir de_caractere, şir_căutat, şir_de_înlocuire)*

- *şir_de caractere* reprezintă şirul de caractere în care se face căutare şi, în cele mai multe cazuri, este numele unei coloane dintr-un tabel, dar poate fi orice expresie care are ca rezultat un şir de caractere.
- *şir_căutat* este un şir format dintr-un caracter sau mai multe, care trebuie căutate în *şir_de caractere.*
- *şir_de înlocuire* este şirul de caractere cu care se înlocuieşte orice apariţie a *şirului_căutat* în *şir_de_caractere.*

Exemplu care înlocuieşte toate liniuţele de despărţire, din numărul de telefon al unei persoane cu puncte:

```
SELECT PERSON_PHONE,
REPLACE(PERSON_PHONE,'-','.')  AS DISPLAY_PHONE FROM PERSON;
```

LTRIM

Funcţia LTRIM elimină spaţiile de la începutul unui şir de caractere. Reţineţi că sunt eliminate *numai* spaţiile de la începutul şirului - cele din mijlocul şi de la sfârşitul şirului sunt lăsate ca atare.

Exemplu general:

```
LTRIM('   String with spaces ')
```

Returnează acest şir: 'String with spaces '

RTRIM

Funcţia RTRIM este similară cu LTRIM, dar elimină spaţiile de la sfârşitul unui şir de

caractere.

Funcţii pentru valori nule (NVL, ISNULL, IFNULL)

Oracle, Microsoft SQL Server şi MySQL pun la dispoziţie o funcţie care înlocuieşte valorile nule cu o valoare specificată. Din nefericire, fiecare implementare foloseşte propriul nume pentru această funcţie: NVL în Oracle, ISNULL în SQL Server şi IFNULL în MySQL. Se pare că în DB2 nu există o funcţie echivalentă.

Exemplu:

```
SELECT IFNULL(LATE_OR_LOSS_FEE, 0) AS LATE_OR_LOSS_FEE FROM
MOVIE_RENTAL WHERE TRANSACTION_ID=9;
```

ASCII

Funcţia ASCII returnează valoarea din setul de caractere ASCII (un număr între 0 şi 255) pentru un şir format dintr-un singur caracter. De exemplu, codul ASCII pentru spaţiu este 32, aşa că ASCII(′ ′) returnează valoarea 32.

CHAR (CHR)

Funcţia CHAR (numită CHR în Oracle şi DB2) returnează caracterul corespunzător unei valori ASCII (un număr între 0 şi 255). De exemplu, funcţia ASCII(44) returnează o virgulă, deoarece codul ASCII pentru virgulă este 44. Această funcţie este utilă în special pentru concatenarea caracterelor care nu pot *fi* afişate sau sunt greu de manipulat în SQL. Câteva dintre caracterele ASCII folosite frecvent cu această funcţie sunt prezentate în tabelul următor. Puteţi folosi funcţia ASCII sau un tabel cu setul de caractere ASCII (uşor de găsit în Internet) dacă vreţi să aflaţi si alte valori.

Valoare ASCII	Caracter
9	Tab
10	Linie nouă
13	Retur de car (CR)
39	Apostrof

Funcţii matematice

Funcţiile matematice returnează rezultatul unei operaţii matematice şi, de obicei, acceptă ca parametru de intrare o expresie numerică, care poate fi o valoare literală, o valoare numerică dintr-o coloană a unui tabel sau orice expresie (inclusiv rezultatul unei alte funcţii) care produce o valoarea numerică.

SIGN

Funcţia SIGN primeşte ca argument o expresie numerică şi returnează una dintre următoarele valori, în funcţie de semnul numărului de intrare.

Valoare	Semnificaţie
-1	nr de intrare este
0	nr de intrare este
1	nr de intrare este
null	valoarea de intrare este nulă

Exemplu:

```
SELECT LATE_OR_LOSS_FEE, SIGN(LATE_OR_LOSS_FEE) AS FEE_SIGN FROM
MOVIE_RENTAL WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

SQRT

Funcţia SQRT primeşte ca argument o expresie numerică şi returnează rădăcina pătrată a acesteia. Sintaxa generală a funcţiei este:

```
SQRT (expresie_numerică)
```

Exemplu:

```
SELECT LATE_OR_LOSS_FEE, SQRT(LATE_OR_LOSS_FEE) AS FEE_SQRT FROM
MOVIE_RENTAL WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

CEILING (CEIL)

Funcţia CEILING returnează cel mai mic întreg mai mare sau egal cu valoarea expresiei numerice furnizată ca parametru de intrare. Cu alte cuvinte, rotunjeşte numărul prin adăugire până la următorul număr întreg. Există câteva probleme interesante de compatibilitate legate de denumire între implementările SQL: Microsoft SQL Server foloseşte numele CEILING, Oracle foloseşte numele CEIL, în timp ce DB2 şi MySQL permite folosirea ambelor nume (CEIL şi CEILING).

Exemplu:

```
SELECT LATE_OR_LOSS_FEE, CEILING(LATE_OR_LOSS_FEE) AS FEE_CEILING
FROM MOVIE_RENTAL WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

FLOOR

Funcţia FLOOR este inversa logică a funcţiei CEILING - returnează cel mai mare întreg mai mic sau egal cu valoarea expresiei numerice furnizată ca parametru de intrare. Cu alte cuvinte, rotunjeşte numărul prin scădere până la următorul număr întreg.

Exemplu:

```
SELECT LATE_OR_LOSS_FEE, FLOOR(LATE_OR_LOSS_FEE) AS FEE_FLOOR
FROM MOVIE_RENTAL WHERE LATE_OR_LOSS_FEE IS NOT NULL;
```

1.3.5. Folosirea avantajelor oferite de vizualizări

O *vizualizare (view)* este o interogare stocată în baza de date, care pune la dispoziţia utilizatorului bazei de date un subset particularizat de date din unul sau mai multe tabele ale bazei de date. Frumuseţea vizualizărilor constă în faptul că după ce sunt create, pot fi interogate ca şi cum ar fi tabele reale. De fapt, nu este nevoie ca utilizatorul să ştie dacă foloseşte o vizualizare sau un tabel real. Mai mult, vizualizările oferă şi alte avantaje:

- Maschează coloanele pe care utilizatorul nu este nevoie sau nu trebuie să le vadă
- Maschează rândurile pe care utilizatorul nu este nevoie sau nu trebuie să le vadă
- Maschează operaţiile complexe, precum uniunile
- Cresc performanţele interogărilor (în unele sisteme RDBMS, cum ar fi Microsoft SQL Server). Sintaxa generală pentru crearea unei vizualizări este:

  CREATE [OR REPLACE] VIEW nume_vizualizare AS interogare_sql;

1.4. ACTUALIZAREA DATELOR FOLOSIND LIMBAJUL SQL

Instrucţiunile DML (Data Manipulation Language), reprezintă o parte a limbajului SQL folosită pentru întreţinerea datelor stocate în tabelele relaţionale ale bazei de date.

Limbajul DML este format din trei comenzi SQL:

- **INSERT** Adaugă noi rânduri într-un tabel al bazei de date
- **UPDATE** Actualizează rândurile existente într-un tabel al bazei de date
- **DELETE** Şterge rânduri dintr-un tabel al bazei de date.

Observaţie: instrucţiunile DML individuale afectează datele dintr-un singur tabel. Atunci când o instrucţiune DML foloseşte o vizualizare, toate coloanele vizualizării referite în instrucţiunea DML trebuie să corespundă unor coloane dintr-un singur tabel fizic al bazei de date.

Instrucţiunile SQL din acest capitol presupun că sistemul DBMS pe care îl utilizaţi foloseşte formatul AAAA-LL-ZZ pentru datele calendaristice.

La formarea instrucţiunilor DML trebuie să ţineţi seama de următoarele aspecte referitoare la restricţiile tabelului modificat:

- ***Restricţii de tip cheie primară.*** Atunci când inseraţi un nou rând într-un tabel, cheia primară a noului rând trebuie să fie unică în întregul tabel. Când modificaţi valoarea unei chei primare (ceea ce se întâmplă rareori), noua valoare trebuie să fie unică în întregul tabel.

- ***Restricţii de unicitate.*** Ca şi în cazul cheilor primare, coloanele pe care a fost definită o restricţie de unicitate trebuie să aibă valori unice în întregul tabel.
- ***Restricţii referenţiale.*** Nu puteţi insera sau actualiza valoarea unei chei externe decât dacă există deja rândul părinte corespondent care conţine valoarea cheii în coloana cheii primare. În sens invers, nu puteţi şterge un rând părinte dacă există rânduri subordonate care referă valoarea din rândul părinte, decât dacă restricţia a fost definită cu opţiunea ON DELETE CASCADE. În general, inserările în tabele trebuie să fie făcute ierarhic (mai întâi rândurile părinte, apoi rândurile copii), iar ştergerile trebuie făcute în ordine inversă (copiii înaintea părinţilor).
- ***Restricţii NOT NULL.*** În cazul instrucţiunilor INSERT, trebuie să specificaţi valori pentru toate coloanele cu restricţii NOT NULL. În cazul instrucţiunilor UPDATE, nu puteţi înlocui valorile unei coloane cu valori nule dacă pe coloana respectivă este definită o restricţie NOT NULL. Dacă instrucţiunea DML referă o vizualizare, nu o puteţi folosi într-o instrucţiune INSERT dacă una dintre coloanele obligatorii ale tabelului (o coloană cu o restricţie NOT NULL) lipseşte din definiţia vizualizării.
- ***Restricţii de verificare (CHECK).*** O instrucţiune INSERT sau UPDATE nu poate stoca într-o coloană o valoare care încalcă o restricţie CHECK definită pentru coloana respectivă.

Instrucţiunea INSERT

Instrucţiunea SQL INSERT este folosită pentru inserarea noilor rânduri de date în tabele. Instrucţiunea are două forme de bază: una în care valorile coloanelor sunt specificate chiar în instrucţiune şi alta în care valorile sunt selectate dintr-un tabel sau o vizualizare, folosind o subinterogare.

Inserarea unui singur rând de date folosind clauza VALUES

Instrucţiunea INSERT care foloseşte o clauză VALUES poate crea un singur rând la fiecare rulare, deoarece valorile pentru rândul de date respectiv sunt specificate chiar în instrucţiune.

Sintaxa generală a instrucţiunii este:

INSERT INTO nume_tabel sau vizualizare [(listă_de_coloane)] VALUES (listă_de_valori);

Reţinem următoarele aspecte:

- Lista de coloane este opţională, dar dacă este inclusă trebuie să fie încadrată în paranteze.
- Dacă lista de coloane este omisă, trebuie specificată o valoare pentru fiecare coloană din tabel, în ordinea în care sunt definite coloanele în tabel. Este bine ca *întotdeauna* să includeţi lista de coloane, deoarece omiterea acesteia face ca instrucţiunea INSERT să fie dependentă de definiţia tabelului. Dacă o coloană este modificată sau în tabel este adăugată o nouă coloană, chiar şi opţională, probabil instrucţiunea INSERT va eşua la următoarea rulare.
- Dacă lista de coloane este specificată, lista de valori trebuie să conţină o valoare pentru fiecare coloană din listă, în aceeaşi ordine. Cu alte cuvinte, între lista de coloane şi lista de valori trebuie să există o corespondenţă unu-la-unu. Orice coloană care lipseşte din listă va primi o valoare nulă, presupunând că valorile nule sunt acceptate în coloana respectivă.
- Cuvântul cheie NULL poate fi folosit în lista de valori pentru specificarea unei valori nule pentru o coloană.

Exemplu care conţine două instrucţiuni INSERT, una care creează un nou rând în tabelul MOVIE, cu o listă opţională de coloane din care este omisă coloana RETAIL_PRICE_VHS, şi alta care creează un rând în tabelul MOVIE_COPY pentru acelaşi timp, dar fără lista opţională de coloane:

```
INSERT INTO MOVIE
(MOVIE_ID, MOVIE_GENRE_CODE, MPAA_RATING_CODE, MOVIE_TITLE,
RETAIL_PRICE_DVD, YEAR_PRODUCED) VALUES (21, 'Drama', 'PG-13', 'Ray', 29.95, '2004');
INSERT INTO MOVIE_COPY VALUES (21, 1, '2005-04-01', null, 'V');
```

Inserări masive folosind o instrucţiune SELECT internă

O altă soluţie, care poate fi folosită pentru a insera rânduri multiple într-un tabel, este

forma care foloseşte o instrucţiune SELECT internă. Această formă este utilă şi pentru stabilirea următoarei valori disponibile pentru o cheie primară cu valori secvenţiale, cum este coloana MOVIE_ID din tabelul MOVIE. De asemenea, poate fi folosită atunci când creaţi un tabel temporar pentru testare şi vreţi să-1 populaţi cu toate datele dintr-un alt tabel.

Sintaxa generală a instrucţiunii este:

INSERT INTO *nume_tabel sau vizualizare* [(*lista__ de__coloane*)] SELECT *instrucţiune_select;*

Reţinem următoarele aspecte:

- Lista de coloane este opţională, dar dacă este inclusă trebuie să fie încadrată în paranteze.
- Dacă lista de coloane este omisă, instrucţiunea SELECT internă trebuie să furnizeze o valoare pentru fiecare coloană din tabel, în ordinea în care sunt definite coloanele în tabel. Este bine ca întotdeauna să includeţi lista de coloane, deoarece omiterea acesteia face ca instrucţiunea INSERT să fie dependentă de definiţia tabelului. Dacă o coloană este modificată sau în tabel este adăugată o nouă coloană, chiar şi opţională, probabil instrucţiunea INSERT va eşua la următoarea rulare.
- Dacă lista de coloane este specificată, instrucţiunea SELECT internă trebuie să furnizeze o valoare pentru fiecare coloană din listă, în aceeaşi ordine. Cu alte cuvinte, între lista de coloane şi setul de rezultate al instrucţiunii SELECT trebuie să există o corespondenţă unu-la-unu. Orice coloană care lipseşte din listă va primi o valoare nulă, presupunând că valorile nule sunt acceptate în coloana respectivă.
- Cuvântul cheie NULL poate fi folosit în instrucţiunea SELECT pentru specificarea unei valori nule pentru o coloană.

Exemplu: să presupunem că toate filmele sunt acum disponibile şi în limba franceză. Tabelul MOVIE_LANGUAGE conţine rânduri pentru limba franceză pentru o parte din filme, aşa că acum vrem să le adăugăm numai pe cele care lipsesc.

```
INSERT INTO MOVIE_LANGUAGE (MOVIE_ID, LANGUAGE_CODE)
SELECT MOVIE_ID, 'fr' FROM MOVIE WHERE MOVIE_ID NOT IN
(SELECT MOVIE_ID FROM MOVIE_LANGUAGE WHERE LANGUAGE_CODE = 'fr');
```

Exemplul următor inserează un nou rând în tabelul MOVIE, folosind instrucţiunea SELECT pentru a găsi valoarea maximă din coloana MOVIE_ID, pe care o incrementează cu o unitate, obţinând astfel cheia primară a noului rând:

```
INSERT INTO MOVIE
(MOVIE_ID, MOVIE_GENRE_CODE, MPAA_RATING_CODE, MOVIE_TITLE,
RETAIL_PRICE_DVD, YEAR_PRODUCED) SELECT MAX(MOVIE_ID)+1, 'Drama', 'PG-13', 'The
Terminal', 24.95, '2004' FROM MOVIE;
```

Instrucţiunea UPDATE

Instrucţiunea UPDATE este folosită pentru actualizarea datelor din coloanele unui tabel (sau ale unei vizualizări).

Sintaxa generală a instrucţiunii UPDATE este:

UPDATE *nume_ tabel sau vizualizare* SET *nume_coloană = expresie*

[, *nume_ coloană = expresie...*] [WHERE *condiţie*];

Reţinem următoarele aspecte:

- Clauza SET conţine o listă cu una sau mai multe coloane, împreună cu o expresie care specifică noua valoare pentru fiecare coloană. În esenţă, aceasta este o listă de perechi nume-valoare, separate prin virgule, cu un operator de egalitate între fiecare nume şi valoare.
- *Expresia* poate fi o constantă, un alt nume de coloană sau orice altă expresie pe care SQL o poate transforma într-o singură valoare, care va fi apoi atribuită coloanei respective.
- Clauza WHERE conţine o expresie care limitează rândurile actualizate. Dacă această clauză este omisă, motorul SQL va încerca să actualizeze toate rândurile din tabel sau din vizualizare.

Exemple:

- Un film adăugat cu MOVIE_ID = 21 a fost introdus cu un preţ incorect. Preţul pentru copia VHS ar trebui să fie 29.95, iar preţul pentru copia DVD ar trebui să fie 34.95. Instrucţiunea următoare actualizează preţurile, introducând valorile corecte. Observaţi că instrucţiunea actualizează două coloane din acelaşi rând al tabelului MOVIE.
  UPDATE MOVIE SET RETAIL_PRICE_VHS = 29.95, RETAIL_PRICE_DVD = 34.95
  WHERE MOVIE_ID = 21;
- Proprietarul magazinului de produse video a decis să crească salariul tuturor funcţionarilor (EMPLOYEE_JOB_CATEGORY = 'C') cu 8 procente. Această instrucţiune va actualiza o singură coloană de pe mai multe rânduri (toate rândurile pentru care corespunde categoria postului). Puteţi folosi un calcul pentru a creşte fiecare salariu cu 8 procente, înmulţind valoarea existentă cu 1.08 şi rotunjind rezultatul la două cifre zecimale, cu ajutorul funcţiei ROUND.
  UPDATE EMPLOYEE SET EMPLOYEE_HOURLY_RATE =
  ROUND(EMPLOYEE_HOURLY_RATE * 1.08, 2) WHERE EMPLOYEE_JOB_CATEGORY = 'C';

Instrucţiunea DELETE

Instrucţiunea DELETE şterge unul sau mai multe rânduri dintr-un tabel. Instrucţiunea poate să folosească şi o vizualizare, dar numai dacă aceasta se bazează pe un singur tabel (ceea ce înseamnă că instrucţiunile DELETE nu pot fi folosite pentru vizualizări care conţin uniuni). În instrucţiunile DELETE nu sunt referite niciodată coloane, deoarece instrucţiunea şterge rânduri întregi de date, inclusiv toate valorile datelor (toate coloanele) din rândurile afectate. Dacă vreţi să ştergeţi o singură valoare din rândurile existente, folosiţi instrucţiunea UPDATE pentru a înlocui valorile respective cu valori nule (presupunând că valorile nule sunt permise în acele coloane). Sintaxa generala a instrucţiunii DELETE este: DELETE FROM *nume_tabel sau vizualizare* [WHERE *condiţie*];

Reţinem următoarele aspecte:

- Clauza WHERE este opţională. Totuşi, este folosită aproape întotdeauna, deoarece o instrucţiune DELETE fără o clauză WHERE încearcă să şteargă *toate râ*ndurile din tabel - ceea ce, în majoritatea cazurilor, nu este rezultatul care se doreşte.
- Atunci când este inclusă, clauza WHERE specifică rândurile care urmează să fie şterse. Orice rând pentru care condiţia WHERE este evaluată ca adevărată este şters din tabel.
- Reţineţi că nu puteţi şterge rânduri dacă încălcaţi astfel o restricţie referenţială. În general, rândurile subordonate trebuie şterse înainte rândurilor părinte.

Exemple:

- Ştergeţi filmul (cu MOVIE_ID = 21). Observaţi că trebuie să ştergeţi mai întâi rândurile corespondente din tabelele MOVIE_COPY şi MOVIE_LANGUAGE, deoarece acestea sunt rânduri „copii" ale rândului din tabelul MOVIE. Dacă încercaţi să ştergeţi mai întâi rândul din tabelul MOVIE, sistemul DBMS va afişa un mesaj de eroare referitor la încălcarea unei restricţii referenţiale.
  DELETE FROM MOVIE_COPY WHERE MOVIE_ID = 21;
  DELETE FROM MOVIE_LANGUAGE WHERE MOVIE_ID = 21;
  DELETE FROM MOVIE WHERE MOVIE_ID = 21;
- Ştergeţi din tabelul MOVIE_LANGUAGE toate rândurile pentru limba spaniolă (LANGUAGE_CODE = 'es'. Reţineţi că în multe implementări SQL se face diferenţierea literelor mari de cele mici, caz în care valoarea codului de limbă trebuie specificată cu litere mici pentru a se potrivi cu datele din tabel.
  DELETE FROM MOVIE_LANGUAGE WHERE LANGUAGE_CODE = 'es';

CAP.2. PROGRAMARE ORIENTATĂ OBIECT ŞI STRUCTURI DE DATE

2.1. CARACTERISTICILE GENERALE ALE PROGRAMĂRII ORIENTATE OBIECT

2.1.1. Principiile ce stau la baza programării orientate obiect

Programarea orientată obiect este unul din cei mai importanţi paşi făcuţi în evoluţia limbajelor de programare spre o mai puternică abstractizare în implementarea programelor. Ea a apărut din necesitatea exprimării problemei într-un mod mai natural fiinţei umane. Astfel unităţile care alcătuiesc un program se apropie mai mult de modul nostru de a gândi decât modul de lucru al calculatorului.

Programarea obiect se desfăşoară ca programarea modulară cu tipuri abstracte de date, în scopul creării posibilităţii de încapsulare a structurilor de date. Ea favorizează reutilizarea şi extensibilitatea aplicaţiilor.

Principalele concepte ce stau la baza programării orientate obiect sunt:

A) Abstractizarea.

Abstractizarea este una din căile fundamentale prin care noi, oamenii, ajungem să înţelegem şi să cuprindem complexitatea, oferind posibilitatea ca un program să ignore unele aspecte ale informaţiei pe care o manipulează, adică posibilitatea de a se concentra asupra esenţialului. Fiecare obiect în sistem are rolul unui "actor" abstract, care poate executa acţiuni, îşi poate modifica şi comunica starea şi poate comunica cu alte obiecte din sistem fără a dezvălui cum au fost implementate acele facilitaţi. Procesele, funcţiile sau metodele pot fi de asemenea abstracte, şi atunci când sunt, sunt necesare o varietate de tehnici pentru a extinde abstractizarea.

O abstracţiune exprimă toate caracteristicile esenţiale ale unui obiect, care fac ca acesta să se distingă de alte obiecte, oferind o definire precisă a graniţelor conceptuale ale obiectului.

- **Comportament vs. Implementare**.

În procesul de abstractizare atenţia este îndreptată exclusiv spre aspectul exterior al obiectului, adică spre comportarea lui, ignorând implementarea acestei comportări. Cu alte cuvinte abstractizarea ne ajuta sa delimitam ferm "**CE face obiectul**" de "**CUM face obiectul ceea ce face"**.

- **Modelul Client-Server. Responsabilităţi. Protocol**.

Comportarea unui obiect se caracterizează printr-o sumă de servicii sau resurse pe care el le pune la dispoziţia altor obiecte. Un asemenea comportament, în care un obiect, numit server, oferă servicii altor obiecte, numite clienţi, este descris de aşa-numitul model client-server.

Totalitatea serviciilor oferite de un obiect server constituie un contract sau o responsabilitate a obiectului faţă de alte obiecte. Responsabilităţile sunt îndeplinite prin intermediul unor operaţii, fiecare operaţie a unui obiect caracterizându-se printr-o semnătură unică, formată din: nume, o listă de parametrii formali şi un tip returnat. Mulţimea operaţiilor unui obiect, împreună cu regulile lor de apelare constituie protocolul obiectului.

B) Încapsularea

Încapsularea este conceptul complementar abstractizării. Dacă rezultatul operaţiei de abstractizare pentru un anumit obiect este identificarea protocolului său, atunci încapsularea are de a face cu selectarea unei implementări şi tratarea acesteia ca pe un secret al respectivei abstracţiuni. **Încapsularea –** numită şi *ascunderea de informaţii*, asigură faptul că obiectele nu pot schimba starea internă a altor obiecte în mod direct (ci doar prin metode puse la dispoziţie de obiectul respectiv); doar metodele proprii ale obiectului pot accesa starea acestuia. Fiecare tip de obiect expune o interfaţă pentru celelalte obiecte care specifică modul cum acele obiecte pot interacţiona cu el.

Prin urmare, încapsularea este procesul în care are loc ascunderea implementării faţă de majoritatea obiectelor-client. Determinarea protocolului pentru un obiect trebuie să preceadă decizia privind implementarea sa. Sintetizând putem defini încapsularea astfel: ***Încapsularea este procesul de compartimentare a elementelor care formează structura şi comportamentul unei abstracţiuni; încapsularea serveşte la separarea interfeţei "contractuale" de implementarea acesteia.***

Interfaţa vs. Implementare

Din definiţia de mai sus rezultă că un obiect este format din două părţi distincte: **interfaţa (protocolul)** şi respectiv **implementarea acestei interfeţe**. Abstractizarea este procesul prin care este definită interfaţa obiectului, în timp ce încapsularea defineşte reprezentarea (structura) obiectului împreună cu implementarea interfeţei. Ascunderea structurii obiectului şi a implementării metodelor sale este ceea ce se înţelege prin noţiunea de ascundere a informaţiei.

Beneficiile încapsulării

- Separarea interfeţei de reprezentarea unui obiect permite modificarea reprezentării fără a afecta în vreun fel pe nici unul din clienţii săi, întrucât aceştia depind de interfaţă şi nu de reprezentarea obiectului-server.
- Încapsularea permite modificarea programelor într-o manieră eficientă, cu un efort limitat şi bine localizat.

Încapsularea aplică principiul abstracţiei: un obiect nu este accesibil pentru operaţiile vizibile (interfaţa sa externă) şi implementarea sa (structurile datelor asociate) este ascunsă. Astfel, modificările datelor rămân locale, la obiecte, fără să afecteze programele utilizatorilor.

Această încapsulare conferă independenţă între programe, operaţii şi date. Un avantaj imediat este acela că diferitele programe de aplicaţii pot fi partajate pe aceleaşi obiecte fără a cunoaşte mecanismul de intrare-ieşire (import/export). Principiul încapsulării poate fi uneori temporar lăsând un acces mai mult sau mai puţin liber datelor dintr-un obiect, făcând posibilă stabilitatea în timp a alegerii implementării datelor.

Accesul direct al datelor la obiecte este un plus de eficienţă care apelează procedura impusă de încapsulare. Între timp când implementarea datelor este modificată (de exemplu se schimbă tipul lor), se va face modificarea în toate programele utilizatoare ale datelor, în mod direct prin programarea structurată. Decizia de încapsulare se va lua făcându-se un compromis între performanţă şi flexibilitate. Putem spune că cele două tipuri de orientări permit răspunsul parţial la problema proiectării structurate.

C) Modularitatea

Scopul descompunerii în module este reducerea costurilor prin posibilitatea de a proiecta şi revizui modulele în mod independent. Clasele şi obiectele obţinute în urma abstractizării şi încapsulării trebuie grupate şi apoi stocate într-o formă fizică, denumită modul.

Modulele pot fi privite ca şi containere fizice în care declarăm clasele şi obiectele rezultate în urma proiectării la nivel logic. Modulele formează aşadar arhitectura fizică a programului.

Modularizarea constă în divizarea programului într-un număr de module care pot fi compilate separat, dar care sunt conectate (cuplate) între ele. Limbajele care suportă conceptul de modul fac în acelaşi timp distincţia între interfaţa modulului şi implementarea sa.

Reguli Generale de Modularizare

1. Structura fiecărui modul trebuie să fie suficient de simplă pentru a putea fi complet înţeleasă.

2. Implementarea unui modul trebuie să depindă doar de interfeţele altor module, respectiv trebuie să fie posibilă modificarea implementării unui modul fără a avea cunoştinţe despre implementarea altor module şi fără a afecta comportarea celorlalte module.

3. Detaliile sistemului care se presupune că se vor modifica independent vor fi plasate în module diferite.

4. Singurele legături între module vor fi acelea a căror modificare este improbabilă.

5. Orice structură de date este încapsulată într-un modul, ea putând fi accesată direct din interiorul modulului dar nu poate fi accesată din afara modului decât prin intermediul obiectelor şi claselor conţinute în acel modul.

Putem defini ***modularitatea ca fiind proprietatea unui sistem care a fost descompus într-un set de module coezive şi slab cuplate***.

D) Ierarhizarea

Abstractizarea este un lucru bun, dar în majoritatea aplicaţiilor - excepţie făcând doar aplicaţiile banale - vom descoperi mai multe abstracţiuni decât putem cuprinde la un moment dat. Încapsularea ne ajută să tratăm această complexitate prin ascunderea interiorului abstracţiunilor noastre. Modularitatea ne ajută şi ea, oferindu-ne o modalitate de a grupa abstracţiuni legate logic între ele. Toate acestea deşi utile, nu sunt suficiente. Adesea un grup de abstracţiuni formează o ierarhie, iar prin identificarea acestor ierarhii, putem simplifica substanţial înţelegerea problemei. Ierarhizarea reprezintă o ordonare a abstracţiunilor.

- ✓ **Polimorfismul**

Este abilitatea de a procesa obiectele diferit în funcţie de tipul sau de clasa lor. Mai exact, este abilitatea de a redefini metode pentru clasele derivate.

- ✓ **Moştenirea (ierarhia de clase)**

Moştenirea – organizează şi facilitează polimorfismul şi încapsularea permiţând definirea şi crearea unor clase specializate plecând de la clase (generale) care sunt deja definite - acestea pot împărtăşi (şi extinde) comportamentul lor fără a fi nevoie de redefinirea aceluiaşi comportament. Aceasta se face de obicei prin gruparea obiectelor în clase şi prin definirea de clase ca extinderi ale unor clase existente. Conceptul de moştenire permite construirea unor clase noi, care păstrează caracteristicile şi comportarea, deci datele şi funcţiile membru, de la una sau mai multe clase definite anterior, numite clase de bază, fiind posibilă redefinirea sau adăugarea unor date şi funcţii noi. O clasă moştenitoare a uneia sau mai multor clase de bază se numeşte clasă derivată.

Moştenirea defineşte o relaţie între clase în care o clasă împărtăşeşte structura şi comportarea definită în una sau mai multe clase (după caz vorbim de moştenire simplă sau multipla). Moştenirea implică o ierarhie de tip generalizare/specializare, în care clasa derivată specializează structura şi comportamentul mai general al clasei din care a fost derivată.

- ✓ **Agregarea (ierarhia de obiecte)**

Agregarea este relaţia între două obiecte în care unul dintre obiecte aparţine celuilalt obiect. Agregarea redă apartenenţa unui obiect la un alt obiect.

2.1.2. Avantajele programării orientate obiect

Orientarea obiect aduce avantaje decisive cum ar fi: *modelarea obiectelor aplicaţiilor*, *modularitatea, reutilizabilitatea şi extensibilitatea codului care conduc la o mai mare productivitate, şi dezvoltarea unei mari calităţi a aplicaţiilor.*

Deoarece utilizatorii doresc ca programele lor să se comporte într-o manieră fiabilă şi previzibilă, este important să se înţeleagă modul în care POO şi condusă de evenimente va contribui la corecta structurare şi modularizare a programului.

Într-o manieră globală, un program orientat obiect este un *ansamblu de obiecte care printr-un schimb de mesaje declanşează anumite operaţii sau metode, facilitându-le stările lor interne şi returnându-le parametrii.*

Prin crearea în mod vizual a unei singure linii de cod, se obţine un program funcţional care, deşi banal, demonstrează toate elementele unei corecte proiectări orientate obiect şi conduse de evenimente.

Obiectele permit reprezentarea în mod direct a entităţilor lumii reale şi a relaţiilor dintre entităţi. Se definesc concepte noi, ca tip, clasă, moştenire, obţinându-se o însumare a avantajelor sistemelor de gestiune a bazelor de date, care interoghează eficient datele, şi a limbajelor procedurale care permit calcule complexe.

În timp ce structura unei părţi din variabile conţine date şi cealaltă funcţii de tratare, se organizează programele în entităţi active compuse din structuri de date ce ascund modul de funcţionare. Acelaşi nume de funcţie poate fi utilizată pentru efectuarea de acţiuni similare la obiecte diferite, ceea ce permite constituirea unui limbaj abstract şi prezentarea în maniere similare a obiectelor înainte diferenţiate.

Avantajele utilizării POO se pot sintetiza astfel:

- **Uşurinţa proiectării şi reutilizării codului**: odată ce este testată corectitudinea funcţionării unor obiecte dintr-o aplicaţie, acestea vor putea fi folosite fără nici o problemă şi în altă aplicaţie. Acest avantaj poate fi valorificat prin constituirea de biblioteci de obiecte. În ceea ce priveşte proiectarea, se facilitează descompunerea problemelor complexe în subprobleme simple, care pot fi uşor modelate cu ajutorul obiectelor.

Abstractizare - proiectanţii pot obţine o imagine de ansamblu, urmărind comportarea obiectelor şi interacţiunile dintre ele, detaliile fiind îngropate în compoziţia obiectelor. Programarea orientată-obiect constituie o bună metodă de organizare a programelor de calcul (software). Proprietăţile POO conduc la un cod principal compact şi elegant. Obiectele pot descrie mai bine conceptele pe care le reprezintă, fiind mai logice şi intuitive decât modul tradiţional, cu simple structuri de date.

- **Siguranţa datelor** - abilitatea obiectelor de a se comporta ca nişte "cutii negre", de a putea fi folosite fără a se cunoaşte compoziţia lor, asigură confidenţialitatea datelor utilizate şi micşorează frecvenţa apariţiilor şi efectul erorilor legate de manipularea greşită a tipurilor de date. Modelând realitatea complexă, tehnicile orientate obiect, pun accentul pe comportamentul datelor, încapsulând în conceptul de obiect atât datele, cât şi operaţiile posibile asupra lor.
- Din punct de vedere **educaţional**, aplicarea POO conduce la formarea rapidă a unor concepte globale de funcţionare a metodei elementului finit. POO permite construcţia unor aplicaţii prin asamblarea unor module existente, la nivel global, reprezentând astfel o metodă modernă, logică şi eficientă nu numai în dezvoltarea de programe, dar şi în utilizarea şi înţelegerea acestora.
- În programarea orientată obiect interfaţa este separată de implementare. Interfaţa este partea vizibilă a clasei, parte care trebuie înţeleasă de utilizatorul acesteia. Implementarea este partea ascunsă, internă clasei, care este importantă doar pentru autorul clasei. Pot exista una sau mai multe implementări pentru o aceeaşi interfaţă. O implementare satisface cerinţele unei interfeţe dacă comportamentul definit de interfaţă este realizat de implementare.

Pe lângă avantajul simplificării, separarea aduce un plus de flexibilitate pentru implementatori, deoarece mai multe implementări pot servi o aceeaşi interfaţă. Implementările pot să difere în ceea ce priveşte eficienţa de timp, spaţiu, preţul sau calitatea documentaţiei puse la dispoziţie, sau orice ale caracteristici non-funcţionale. De asemenea o singură implementare poate să satisfacă mai multe interfeţe. În acest caz implementarea conţine o uniune de metode cerute de fiecare din interfeţe.

Numeroase limbaje de programare includ tehnici POO. În lucrarea de faţă se foloseşte, pentru exemplificare, limbajul Visual C++, deoarece C++ reprezintă limbajul standard în lumea POO. C++ permite ca programarea procedurală şi cea orientată-obiect să fie folosite împreună într-un limbaj destul de uniform.

Limbajul C++ prezintă şi alte avantaje faţă de celelalte limbaje obiectuale, cum ar fi:

√ C++ produce un cod cu un timp de execuţie foarte eficient;
√ existenţa multor compilatoare, inclusiv a celor gratuite;
√ posibilitatea utilizării modulelor existente Fortran şi C;
√ legăturile cu sistemul de operare Unix (datorate limbajului C);

√ larga utilizare şi acceptare a limbajului C++ a produs numeroase implementări, pe platforme diferite, existând multe biblioteci şi programe de calcul reutilizabile;
√ posibilitatea de interfaţare cu numeroase programe importante.

Caracteristici generale ale claselor în Visual C++

2.1.3.1. Conceptul de clasă şi tipuri de clase

O clasă este schema, planul după care este creat un obiect. În interiorul unei clase sunt precizate *denumirile tuturor variabilelor şi tipurile de valori pe care acestea le vor stoca* (numeric, şir de caractere, etc.), *alături de toate operaţiile* (metodele) *pe care un obiect, construit din respectiva clasă, le va conţine*.

Clasa reprezintă *o abstractizarea a elementelor (proprietăţilor şi operaţiilor) comune partajate de o mulţime de obiecte şi descrie implementarea acestora*.

Folosind instrumentele moderne precum **Visual C++,** alcătuirea unui program este cu mult mai simplă. Mediul Visual **C++** permite editarea, compilarea, executarea şi depanarea unui program în interiorul aceluiaşi mediu.

Programarea orientată-obiect presupune ***ordonarea obiectelor şi a acţiunilor din lumea reală sub formă de clase care pot fi create, manevrate şi distruse***.

Datele care alcătuiesc un obiect şi funcţiile exercitate asupra acelui obiect sunt combinate pentru a forma o clasă, sau o descriere a acelui obiect.

Clasele pot moşteni funcţionalitatea altor obiecte, putând fi cu uşurinţă adăugate clase noi care depăşesc clasele existente.

O clasă trebuie să asigure o interfaţă, utilizată pentru manipularea obiectelor existente în acea clasă. Pe cât posibil, detaliile de implementare trebuie ascunse utilizatorului, ceea ce permite programatorului dotarea claselor complicate cu o interfaţă simplă, precum şi modificarea detaliilor de implementare, dacă este necesar.

Deoarece structurile orientate obiect descriu integral clasa din care face parte obiectul respectiv, fiecare clasă poate fi reutilizată cu uşurinţă, întrucât datele şi funcţiile descrise de clasa respectivă sunt integrate.

Deoarece implementarea unei clase poate fi ascunsă în spatele unei interfeţe, detaliile de implementare ale respectivei clase sunt uşor de modificat fără a-i afecta pe utilizatorii acesteia, atâta timp cât interfaţa nu se modifică.

Atunci când se utilizează limbajul Visual **C++**, obiectele sunt descrise printr-o **clasă**. O clasă are două componente principale:

- **declaraţia clasei** - conţine interfaţa clasei şi informaţii privind datele membre ale respectivei clase. Interfaţa clasei este de obicei localizată într-un fişier antet cu extensia **.h**. Orice fişier din program care foloseşte clasa respectivă trebuie să folosească directiva **#include**, pentru ca declaraţia de clasă să fie adăugată fişierului sursă de către preprocesor;
- **implementarea clasei** - include toate funcţiile membru care au fost declarate ca parte a clasei. De obicei implementarea clasei este localizată într-un fişier având extensia **.CPP**.

Pentru a face clasele uşor de citit şi de întreţinut, se va descompune codul asociat în două tipuri de fişiere: *fişiere header* şi *fişiere de cod*. Fişierele din prima categorie conţin *declaraţiile atributelor* şi *funcţiilor membru clasei* în cauză. Aceasta nu conţine altceva decât structura clasei. Fişierele din a doua categorie conţinând *codul* sau *implementarea* nu sunt altceva decât fişiere ASCII care conţin definiţia fiecărei metode declarate în fişierele header corespunzătoare.

Fişierul header trebuie să conţină următoarele informaţii:

- numele fişierului trebuie să reprezinte clasa;
- descriere a ceea ce face clasa;
- structura clasei, cea care este implementată în porţiunea privată;
- funcţiile membru împărţite pe categorii;

- comentarii tactice şi strategice ale funcţiilor membru şi grupurile de funcţii membru, chiar şi de atribute.

Funcţiile membru trebuie să fie documentate în fişierele header prin aşa numitele **comentarii tactice** (comentarii conţinute într-o singură linie ce sunt utile noilor utilizatori ai clasei în scopul înţelegerii acesteia). **Comentariile strategice** sunt utilizate în documentarea grupurilor sau blocurilor de funcţii aflate într-o anume legătură unele cu altele.

Fiecare funcţie membru declarată în fişierul header trebuie să-şi găsească implementarea corespunzătoare în fişierul de cod asociat clasei. Mai mult, fişierul de cod este amplasat acolo unde detaliile de modificare ale programului pot fi documentate.

C++ permite definirea de funcţii având acelaşi nume, dar tipuri şi număr diferite de parametri. Tipul valorilor returnate de astfel de funcţii este irelevant în identificarea variantei apelate. Aceasta este o formă ad-hoc de polimorfism, cunoscută sub denumirea de **redefinirea funcţiilor**.

O altă formă de polimorfism acceptată în **C++** este **redefinirea operatorilor**. De fapt, aceasta nu este altceva decât o variantă restrânsă a primei forme de polimorfism ad-hoc. Prin această metodă, este posibilă redefinirea operatorilor în vederea operării acestora asupra tipurilor de date definite de utilizator, fiind o soluţie deosebit de importantă în toate aplicaţiile cu un bogat suport matematic, în special.

Redefinirea funcţiilor şi operatorilor poate creşte gradul de înţelegere a codului. Totuşi, este important să nu se uite că utilizarea excesivă a acestei tehnici poate conduce la obţinerea unui cod mult mai criptat decât este cazul. În cazul redefinirii funcţiilor, se vor utiliza nume care să aibă sens în cadrul problemei.

Privitor la diferitele clase care pot apărea într-un program, există două modalităţi de organizare a claselor:

√ **ierarhii** - o ierarhie poate fi folosită pentru organizarea claselor după modelul unui arbore genealogic, cu clase care moştenesc datele şi funcţiile de la predecesorii lor;

√ **categorii** - adesea, un set de clase se folosesc împreună şi prezintă un fel de dependenţă sau o altă categorie de relaţii între ele. Aceste clase pot fi organizate într-o categorie, pentru a simplifica utilizarea lor.

În majoritatea limbajelor orientate obiect, clasele pot moşteni proprietăţile unei **clase de bază**. O clasă de bază are anumite proprietăţi generale care sunt incluse în clasele derivate din acestea.

Organizarea claselor într-o ierarhie înainte de începerea scrierii declaraţiilor de clasă este o modalitate de a căuta punctele comune ale claselor. Prin organizarea claselor, se pot determina cu uşurinţă relaţiile care există între toate clasele, ceea ce poate contribui la selectarea clasei de bază şi la determinarea claselor care trebuie create.

În multe cazuri, între clasele care lucrează împreună, există o strânsă interdependenţă. Clasele între care există legături strânse sunt adesea grupate în categorii care sunt frecvent folosite în grup; în multe cazuri nu are nici un rost utilizarea unei clase care este componentă a unui grup mai mare.

C++ suportă atât moştenirea simplă cât şi pe cea multiplă, însă dintre cele două, cea simplă este cea mai utilizată în cadrul proiectelor **C++**, aplicarea moştenirii multiple fiind mult mai dificilă. Utilizarea moştenirii în **C++** poate ajuta la creşterea productivităţii, dacă este aplicată corespunzător.

Clasele identificate drept clase predecesor, sau clase de bază, conţin date şi funcţii membru care sunt incluse în toate clasele descendent, sau derivate.

Când se foloseşte conceptul de moştenire, o clasă de bază stabileşte un fel de contract care trebuie respectat de către toate subclasele. Dacă o operaţie poate fi executată utilizând o clasă de bază, toate subclasele trebuie să fie capabile să execute acea operaţie. Cu alte cuvinte, *într-o structură cu adevărat orientată-obiect se impune posibilitatea înlocuirii unei subclase cu o clasă de bază*. Acesta este cunoscut sub numele de **principiul de înlocuire al lui Liskov**.

Unul din obiectivele moştenirii este acela de a partaja un cod între mai multe clase, ceea ce nu face altceva decât să crească factorul de reutilizare al codului. În general, este bine de evitat ierarhiile de moştenire stufoase.

Principiile de ascundere a informaţiei şi încapsulare sunt suportate de Visual **C++**. Acest limbaj implementează aceste două noţiuni prin intermediul cuvintelor cheie **public** şi **private**. Astfel, membrii clasei care sunt declaraţi în *secţiunea privată* sunt vizibili numai instanţelor şi membrilor clasei, fiind în acelaşi timp inaccesibili celorlalte clase şi instanţe ale acestora.

Secţiunea privată este utilizată în primul rând pentru informaţiile stării unui obiect. Ea conţine declaraţii ale datelor ce reprezintă atributele unui obiect. Această secţiune poate conţine, de asemenea, declaraţiile funcţiilor membru care nu sunt oferite spre utilizare clienţilor clasei.

Membrii claselor care sunt declaraţi în *secţiunea publică* a acesteia sunt accesibili tuturor obiectelor şi funcţiilor, secţiunea publică constituind implementarea serviciilor pe care o clasă le oferă clienţilor săi.

Pentru apelul funcţiilor membre publice sau pentru accesul la datele publice ale unui obiect, din funcţii care nu sunt membre, se folosesc operatorii de selecţie (.) şi (->), ca în cazul structurilor şi uniunilor din C.

Un caz special al paradigmei client/server ne este oferit de **C++**, respectiv cazul *relaţiei dintre o clasă dată şi derivatele sale*, unde este posibil să se definească zone protejate. Membrii acestor zone se comportă ca membrii publici în clasele derivate, astfel aceste elemente violează principiul de ascundere a informaţiei.

Numele tuturor funcţiilor membru sunt prefixate de numele de clasă ale acestora, aceasta fiind unica modalitate prin care compilatorul poate determina clasa căreia îi aparţine funcţia respectivă. Unica excepţie de la regulă o constituie o funcţie definită în interiorul declaraţiei de clasă.

Constructori şi destructori

În cazul variabilelor obişnuite, compilatorul asigură alocarea spaţiului de memorie şi eventual iniţializarea explicită cu valori iniţiale în declaraţie. Pentru variabilele dinamice, compilatorul C nu dispune de nici o metodă de iniţializare şi nici operatorul **new** nu rezolvă toate situaţiile. În acest caz, rămâne în grija programatorului atribuirea de valori adecvate datelor înainte de utilizare. Această abordare este nesatisfăcătoare în multe situaţii în cazul obiectelor.

Pentru crearea, iniţializarea, copierea şi respectiv distrugerea obiectelor, în C++ se folosesc funcţii speciale, numite constructori şi destructori.

Un **constructor**, numit uneori şi ctor, este o funcţie membru specială, creată la crearea unui obiect al unei clase. Constructorul se apelează automat la crearea fiecărui obiect al clasei, static, automatic sau dinamic (cu operatorul new), inclusiv pentru obiecte temporare. Un obiect se creează atunci când este definit, sau când este creat din punct de vedere dinamic.

Utilitatea constructorilor este evidentă cel puţin sub două aspecte:

- constructorul asigură iniţializarea corectă a tuturor variabilelor membru ale unui obiect;
- constructorul oferă o garanţie că iniţializarea unui obiect se va realiza exact o dată.

De multe ori este util să existe mai multe moduri de iniţializare a obiectelor unei clase. Acest lucru se poate realiza furnizând diferiţi constructori. Atâta timp cât constructorii diferă suficient în tipurile argumentelor lor, compilatorul le poate selecta corect, unul pentru fiecare utilizare.

Un **destructor**, cunoscut şi sub numele de dtor, este o funcţie membru specială care este apelată la distrugerea unui obiect. Destructorul este declarat ca neavând tip de valoare returnată şi nu este declarat niciodată cu o listă de parametrii. Numele destructorului este alcătuit din numele clasei, precedat de un caracter tilda (~). Nu este absolut necesară definirea unui destructor, decât dacă există operaţii specifice care trebuie efectuate pentru a face curăţenie, cum ar fi eliberarea resurselor unui sistem care au fost alocate.

Destructorul este apelat automat la eliminarea unui obiect, la încheierea timpului sau de viata, sau poate fi solicitat prin program, cu operatorul delete.

Constructorii şi destructorii se declară şi se definesc similar cu celelalte funcţii membre, dar se disting de acestea printr-o serie de caracteristici specifice:

- numele funcţiilor constructor sau destructor coincide cu numele clasei căreia îi aparţin;
- în declaraţie şi definiţie nu se specifică nici un tip de rezultat, nici măcar void;
- constructorii pot avea parametrii, inclusiv parametrii impliciţi, şi pot fi supradefiniţi; destructorii nu au aceste proprietăţi;
- nu se pot utiliza pointeri către constructori sau destructori;
- dacă o clasă nu dispune de constructori şi destructori definiţi, compilatorul va genera automat un constructor implicit, respectiv un destructor, funcţii publice.
- constructorii sunt lansaţi în ordinea declarării obiectelor iar destructorii în ordine inversă.

Altă categorie de constructor este constructorul de copiere. Necesitatea de copiere a obiectelor intervine când un obiect este transferat ca parametru sau rezultat al unei funcţii, sau la crearea unui obiect temporar. Soluţia oferită de C++ constă în utilizarea unui constructor special, numit constructor de copiere. În absenţa unei definiţii explicite în cadrul clasei, compilatorul generează automat un constructor de copiere care iniţializează datele noului obiect cu valorile corespunzătoare din obiectul specificat, prin copiere membru cu membru. Aceasta nu este o soluţie bună în cazul în care clasa conţine membrii variabile dinamice, caz în care este necesară scrierea unui constructor de copiere special. Declaraţia constructorului de copiere pentru o clasă trebuie să specifice un parametru unic de tipul referinţă de obiecte a acelei clase.

Supraîncărcarea funcţiilor membre şi a operatorilor

Funcţiile membre ale unei clase pot fi supraîncărcate. Mecanismul nu diferă de cel al supraîncărcării funcţiilor independente.

Programele manipulează obiecte care sunt reprezentări concrete ale unor concepte abstracte. De exemplu, datele de tip "int" din C++, împreună cu operatorii +, - , *, /, etc., furnizează o implementare restrictivă a conceptului matematic de întregi. Astfel de concepte includ de obicei un set de operatori care reprezintă operaţiile de bază asupra obiectelor.

Clasele furnizează o facilitate de reprezentare a obiectelor neprimitive împreună cu un set de operaţii care pot fi efectuate cu astfel de obiecte. Se permite programatorului să furnizeze o notaţie mai convenţională şi mai convenabilă pentru a manipula obiectele unei clase. Acest lucru se realizează prin *supraîncărcarea operatorilor*. Trebuie evidenţiat faptul că operatorii sunt deja supraîncărcaţi, fie pentru a putea opera asupra mai multor tipuri de bază (de ex. operatorii aritmetici admit ca operand orice tip numeric), fie pentru a efectua mai multe operaţii matematice (de ex. * este asociat înmulţirii dar şi referirii datelor de tip pointer). Operaţia adecvată este selectată de compilator în funcţie de tipul operanzilor.

Procedeul de supraîncărcare constă în definirea unei funcţii cu numele operator simbol unde:

- operator este cuvântul cheie dedicat
- simbol este simbolul oricărui operator C++, mai puţin următorii: . (punct) .* :: ?:

intaxa supraîncărcării unui operator :

```
TipNumeClasa::operatorNumeOperator (ListaArgumente)
 {
     BlocInstructiuni
 }
```

Funcţii virtuale şi clase de bază abstracte

Cuvântul cheie virtual este utilizat în header-ul unei clase de bază în vederea declarării funcţiilor care definesc comportamentul tuturor claselor derivate din această clasă de bază. O funcţie membru virtuală trebuie să posede un corp, acesta situându-se în fişierul de cod asociat clasei. Acest mecanism permite implementarea polimorfismului.

Când se foloseşte o funcţie virtuală, compilatorul construieşte un tabel special, denumit **tabel de funcţii virtuale** (virtual function table) sau vtabl, care se foloseşte pentru a păstra evidenţa funcţiilor care trebuie apelate pentru fiecare obiect din clasa respectivă. Când este apelată o funcţie virtuală, vtab-ul este folosit pentru accesarea indirectă a funcţiei corecte.

Suprasarcina (overhead) adăugată prin utilizarea tabelului de funcţii virtuale este destul de redusă, dar poate deveni semnificativă dacă se dispune de mii de obiecte mici sau dacă viteza de execuţie este hotărâtoare. Din acest motiv, o funcţie trebuie precizată ca fiind virtuală, deoarece acest caracter nu este prestabilit.

Principalul, avantaj al utilizării funcţiilor virtuale poate fi observat în momentul rulării, sistemul putând determina tipul obiectului din care provine funcţia membru apelată.

O funcţie virtuală pură este o funcţie virtuală fără corp. Aceste funcţii sunt declarate în clasele de bază. Derivarea unei clase dintr-o clasă de bază care conţine cel puţin o funcţie virtuală pură presupune declararea şi definirea acestor funcţii în cadrul clasei derivate, în caz contrar producându-se o eroare de compilare.

Utilizarea acestor funcţii autorizează presupunerea că o clasă de bază a fost declarată ca având un comportament normal dar trebuie implementat în toate cazurile speciale. Este nevoie de ceva mult mai puternic pentru a asigura implementarea funcţiilor de către clienţi (în acest caz clase derivate).

Scrierea de funcţii virtuale pure în clasa de bază este echivalentă cu adoptarea unei asigurări: clasa de bază ştie că viitorii săi clienţi trebuie să implementeze aceste funcţii, iar clienţii sunt obligaţi să o facă.

O clasă de bază abstractă (CAB) este o clasă care are cel puţin o funcţie membru virtuală pură. Clasele de bază abstracte nu pot avea instanţe.

Întotdeauna se va încerca să se găsească cât mai multe clase abstracte atunci când se proiectează un produs software utilizând tehnicile orientate obiect. O clasă de bază abstractă apare dintr-un set de clase care partajează atribute şi/sau comportament util. Dacă se descoperă un comportament care este partajat de un număr de clase, va trebui proiectată o superclasă pentru capturarea acestuia într-un singur loc.

Un alt avantaj al claselor abstracte de bază îl constituie faptul că pot ţine locul oricăreia din clasele derivate. Cu alte cuvinte, având o colecţie de clase abstracte de bază, este mult mai uşor să se clasifice clasele, deoarece majoritatea noilor clase pot fi considerate ca fiind derivate ale cel puţin unei clase abstracte.

Este posibilă derivarea de clase atât din cele abstracte cât şi din cele concrete. Combinarea corectă a celor două tipuri de clase va permite obţinerea **ierarhiilor de clase C++**. Regulile de bază sunt:

- alegerea unei clase abstracte de bază pentru a servi drept clasă de bază tuturor celorlalte clase (abstracte sau nu). Clasele abstracte de bază vor deriva direct din aceasta;
- este posibilă crearea unei clase de bază abstracte prin derivarea dintr-o altă clasă de bază abstractă;
- clasele concrete sunt proiectate pentru a fi instanţiate. Acestea pot proveni din derivare, fie dintr-o clasă abstractă fie dintr-una concretă;
- niciodată nu se va deriva o clasă abstractă dintr-una concretă.

Polimorfismul

Polimorfismul este cel mai adesea folosit în programarea orientată obiect pentru a desemna funcţiile sau operatorii care pot fi aplicaţi mai multor categorii diferite de obiecte. Un aspect al polimorfismului este capacitatea utilizării unei funcţii în mai multe obiecte diferite, fără a fi necesar un nume nou de funcţie pentru fiecare obiect.

Limbajul Visual **C++** permite "supraîncărcarea" numelui unei funcţii, ceea ce înseamnă că aceasta poate avea mai multe definiţii. Compilatorul determină funcţia care trebuie apelată în funcţie de lista de parametri.

Domeniu

Domeniul unei variabile se referă la vizibilitatea acesteia într-un program. Un identificator folosit pentru a denumi o variabilă sau o funcţie, are un anumit domeniu atunci când este creat, iar acest domeniu determină modul şi locul în care poate fi folosit respectivul nume. Dacă o variabilă se află în domeniu într-un anume punct din program, atunci este vizibilă şi poate fi folosită în majoritatea cazurilor. Dacă variabila se află în afara domeniului atunci nu este vizibilă şi programul nu o va putea folosi.

În general, un program trebuie să folosească variabile al căror domeniu să fie cât mai redus posibil. Cu cât domeniul este mai restrâns, cu atât sunt mai mici şansele unei utilizări accidentale a unui identificator sau expunerii acestuia la efecte colaterale. De exemplu, întotdeauna este mai indicat transferul obiectelor ca parametrii ai unei funcţii decât utilizarea variabilelor globale. Utilizarea variabilelor locale pentru o anumită funcţie contribuie la capacitatea de reutilizare a funcţiei şi elimină problemele de sincronizare într-un mediu cu execuţii simultane cum este Windows.

Domeniul de definiţie al unui nume prin care se identifică o entitate a unui program este zona din program în care numele este cunoscut şi poate fi folosit. Dat fiind că un nume este făcut cunoscut printr-o declaraţie, domeniile numelor se diferenţiază în funcţie de locul în care este introdusă declaraţia în program. Un nume declarat într-un bloc este *local* blocului şi poate fi folosit numai în acel bloc, începând din locul declaraţiei şi până la sfârşitul blocului şi în toate blocurile incluse după punctul de declaraţie.

Argumentele formale ale unei funcţii sunt tratate ca şi când ar fi declarate în blocul cel mai exterior al funcţiei respective. Un nume declarat în afara oricărui bloc are ca domeniu fişierul în care a fost declarat şi poate fi utilizat din punctul declaraţiei până la sfârşitul fişierului. Numele cu domeniu fişier se numesc nume *globale*.

Un nume este vizibil în întregul său domeniu de definiţie dacă nu este redefinit într-un bloc inclus în domeniul respectiv. Dacă într-un bloc interior domeniului unui nume se redefineşte (sau se redeclară) acelaşi nume, atunci numele iniţial (din blocul exterior) este parţial ascuns, şi anume în tot domeniul redeclarării. Un nume cu domeniu fişier poate fi accesat într-un domeniu în care este ascuns prin redefinire, dacă se foloseşte operatorul de rezoluţie pentru nume globale (::). Redefinirea unui nume este admisă numai în domenii diferite. Dacă unul din domenii este inclus în celălalt domeniu, atunci redefinirea provoacă ascunderea numelui din domeniul exterior în domeniul interior. Redefinirea în domenii identice produce eroare de compilare.

O entitate este creată atunci când se întâlneşte definiţia sa (care este unică) şi este distrus (în mod automat) când se părăseşte domeniul ei de definiţie. O entitate cu nume global se creează şi se iniţializează o singură dată şi are durata de viaţă până la terminarea programului. Entităţile locale se creează de fiecare dată când execuţia programului ajunge în punctul de definire a acestora, cu excepţia variabilelor locale declarate de tip static, care se iniţializează o singură dată la prima execuţie a instrucţiunii. O variabilă globală sau statică neiniţializată explicit, este iniţializată automat cu 0.

Domeniul unui identificator intră în acţiune la fiecare declarare a unui identificator. Tipurile de domenii disponibile într-un program **C++** sunt următoarele:

- **Domeniu local**

Există mai multe categorii de domenii locale, cel mai simplu exemplu de domeniu local reprezentându-l o variabilă sau un alt obiect care este declarat în exteriorul oricărei funcţii. Când variabila se află în domeniu, de la punctul în care a fost declarată şi până la finalul fişierului sursă, acest tip de domeniu local este denumit **domeniu fişier**. Toate declaraţiile care survin în exteriorul definiţiilor de clase sau funcţii au acest tip de domeniu.

Variabilele declarate în interiorul corpului unei funcţii au domeniu local. Acestea sunt vizibile numai în interiorul funcţiei.

O altă categorie de domenii locale este **domeniul bloc**, unde o variabilă din interiorul unei instrucţiuni compuse sau al unui alt bloc este vizibilă până la capătul blocului. Variabilele

declarate în interiorul instrucţiunilor condiţionale au de asemenea domeniu tip bloc.

- **Domeniu tip prototip de funcţie - a**cest domeniu există numai în interiorul unui prototip de funcţie, fiind extrem de limitat. O variabilă cu astfel de domeniu, este în realitate numai o variabilă definită într-un prototip de funcţie, al cărei domeniu se încheie la finalul prototipului.
- **Domeniu tip funcţie -** acest tip de domeniu este rar întâlnit. Conceptul care a stat la baza sa este desemnarea etichetelor declarate în interiorul definiţiei unei funcţii. Limbajul C++ nu acceptă saltul la o etichetă situată în exteriorul funcţiei curente şi respectiv că acelaşi nume de etichete pot fi folosite în mai multe funcţii.
- **Domeniu tip clasă -** numele unui membru tip clasă, uniune sau structură este strâns asociat cu clasa respectivă şi are domeniu tip clasă. Un identificator cu domeniu tip clasă poate fi folosit în interiorul clasei, uniunii sau structurii. Dacă numele unei clase sau al unei variabile este utilizat pentru a controla accesul la un identificator, atunci acesta este vizibil şi în afara clasei.

Durata de viaţă a unui identificator

Într-un program Visual **C++**, fiecare variabilă sau obiect are o anumită durată de viaţă, care este separată de vizibilitatea sa. Astfel, programatorul are posibilitatea controlării momentului creării şi a distrugerii unei variabile.

Durata de viaţă a unui identificator este foarte importantă în conceperea unui program. Obiectele de dimensiuni mari pot fi costisitor de creat, dar şi de distrus.

O variabilă poate fi ascunsă de către altă variabilă având acelaşi nume, dar nu un alt domeniu, ceea ce înseamnă că este posibil ca o variabilă să existe şi în afara domeniului propriu. Aceasta se întâmplă deseori atunci când numele unei variabile este folosit în domenii tip bloc sau clasă şi maschează o altă variabilă care foloseşte aceleaşi nume.

O variabilă declarată static într-o funcţie este creată la începutul programului şi nu este distrusă înainte de încheierea acestuia. Acest fapt este util când se doreşte ca variabila sau obiectul să-şi amintească valoarea între două apelări ale funcţiei.

2.1.3.2. Clasele de bază MFC

Biblioteca de clase MFC poate fi împărţită pe trei nivele. La primul nivel se află ***clasele care descriu comportamentul obiectelor grafice şi încapsulează funcţiile API*** (Application Program Interface).

Aceste obiecte (cum ar fi de exemplu ferestrele, butoanele, fonturile, contextul dispozitiv, etc.) sunt create şi accesate de către programatorii Windows prin intermediul unor funcţii sistem. Prin urmare clasele din MFC de la acest nivel încapsulează practic obiectele şi funcţiile care le sunt ataşate existente în API Windows, oferind un grad ridicat de abstractizare.

La al doilea nivel se află ***clasele care nu depind de sistemul de operare Windows şi care implementează structuri fundamentale de date*** (liste, tablouri, dicţionare, şiruri de caractere, dată calendaristică etc.).

Un mediu de lucru al aplicaţiilor este o colecţie de componente soft (clase, macro-uri, funcţii globale) care pot fi utilizate în realizarea unei aplicaţii generice.

Biblioteca de clase MFC include un mare număr de clase corespunzătoare programării în Windows. Majoritatea acestor clase sunt derivate din CObject, clasă care se află la rădăcina ierarhiei de clase din MFC. Suplimentar, orice clasă care reprezintă o fereastră sau un control este derivată din clasa CWnd, care manevrează funcţiile de bază comune tuturor ferestrelor.

Clasele CObject şi CWnd utilizează funcţiile virtuale, care permit programului să acceseze funcţii de uz general, prin intermediul unui pointer de bază. Aceasta permite utilizarea cu uşurinţă a oricărui obiect derivat din CObject sau CWnd la interacţiunea cu cadrul de lucru MFC.

☐ ***Clasa de bază CObject***

Aproape orice clasă utilizată într-un program MFC este derivată din CObject. Clasa

CObject asigură patru tipuri de servicii:

- **diagnosticarea gestiunii memoriei furnizează mesaje de diagnostic la detectarea unor pierderi de memorie** (memory leak). Aceste pierderi sunt adeseori provocate de eşuarea operaţiei de eliberare a unor obiecte create dinamic;
- **suportul creării dinamice foloseşte clasa CRuntimeClass pentru a permite crearea obiectelor la rularea programului**. Această operaţie este diferită de crearea dinamică a obiectelor utilizând operatorul new;
- **suportul de serializare permite stocarea şi încărcarea unui obiect într-o modalitate orientată-obiect**;
- **informaţia de clasă runtime** (în timpul execuţiei). Biblioteca de clase MFC foloseşte informaţia de clasă runtime pentru a oferi informaţie de diagnosticare la detectarea erorilor în program sau la serializarea obiectelor în sau din spaţiul de stocare.

☐ ***Clasa de bază CWnd***

Această clasă, este derivată din CObject şi oferă un grad mare de funcţionalitate, propriu tuturor ferestrelor dintr-un program MFC. Sunt incluse aici şi casetele de dialog şi controalele, care nu sunt decât versiuni specializate de ferestre. Clasa CWnd defineşte funcţii care pot fi aplicate oricărui obiect din CWnd, inclusiv acelor obiecte care sunt exemplare ale claselor derivate din CWnd.

Clasele de bază CObject şi CWnd sunt două exemple de clase de bază care se utilizează pentru a asigura funcţiile esenţiale ale unui mare număr de clase prin intermediul unei clase de bază.

Aproape fiecare obiect semnificativ dintr-un program MFC este un exemplar din clasa CObject. Aceasta permite utilizarea suportului MFC pentru descoperirea multor pierderi frecvente de memorie şi a altor tipuri de erori de programare. De asemenea clasa CObject declară funcţii care pot fi utilizate pentru a oferi diagnosticări în timpul rulării şi suport pentru serializare.

Fiecare fereastră dintr-un program MFC este un obiect CWnd. Clasa CWnd derivă din clasa CObject, deci are încorporate toate funcţiile acesteia. Utilizarea clasei CWnd pentru manevrarea controalelor şi a ferestrelor din program permite programatorului să beneficieze de avantajele polimorfismului: clasa CWnd **asigură toate funcţiile generale de fereastră pentru toate tipurile de ferestre**. Aceasta înseamnă că în multe cazuri nici nu este necesară cunoaşterea exactă a tipului controlului sau ferestrei accesate prin intermediul unui pointer CWnd.

Clasele CObject şi CWnd sunt folosite în diverse moduri. Clasa CObject este folosită în mod normal drept clasă de bază atunci când programatorul îşi creează propriile clase. Clasa CWnd este adeseori transferată drept parametru de funcţie şi este folosită ca pointer generic pentru orice tip de fereastră într-un program MFC.

Când este folosită drept clasă de bază CObject asigură o mare parte din funcţiile de bază pentru o altă clasă. Se poate controla numărul acestor funcţii asigurate de CObject utilizând funcţii macro în declaraţia clasei derivate şi în fişierele de definire.

Există patru nivele de asistenţă asigurate de clasa CObject claselor sale derivate:

√ **nivelul de bază**, cu detecţia pierderilor de memorie, nu necesită funcţii macro;

√ **asistenţa pentru identificarea claselor la rulare** necesită utilizarea funcţiei macro DECLARE_DYNAMIC în declaraţia clasei şi a funcţiei IMPLEMENT_DYNAMIC în definiţia clasei;

√ **asistenţa pentru crearea dinamică a obiectelor** necesită utilizarea funcţiei macro DECLARE_DYNCREATE în declaraţia clasei şi a funcţiei macro IMPLEMENT_DYNCREATE în definiţia clasei;

√ **asistenţa de serializare** necesită utilizarea funcţiei macro DECLARE_SERIAL în declaraţia clasei şi a funcţiei macro IMPLEMENT_SERIAL în definiţia clasei.

Toate funcţiile macro ale clasei CObject au un mod de utilizare asemănător. Toate funcţiile macro de tip DECLARE au un singur parametru, numele clasei. Funcţiile macro de tip

IMPLEMENT necesită, în general doi parametrii: numele clasei şi numele clasei de bază imediat superioare. Funcţia IMPLEMENT_SERIAL reprezintă o excepţie, deoarece necesită trei parametrii.

Există două moduri de creare dinamică a obiectelor. Prima metodă, utilizează operatorul **C++** new pentru a aloca dinamic un obiect din spaţiul de alocare disponibil. O a doua metodă este folosită în special în cadrul MFC şi foloseşte o clasă specială CRuntimeClass, şi funcţia macro RUNTIME_CLASS. Clasa CRuntimeClass se poate folosi pentru a **determina tipul unui obiect** sau pentru **crearea unui obiect nou**.

Biblioteca de clase MFC oferă mai multe facilităţi de diagnosticare, majoritatea aflându-se sub forma unor funcţii macro folosite numai într-o versiune de depanare a programului, care oferă posibilitatea de a beneficia de avantajele ambelor situaţii (rulare şi depanare). În faza de dezvoltare şi de testare a programului, se pot folosi funcţiile de diagnosticare MFC pentru a verifica existenţa unui număr minim de erori, deşi programul rulează cu o suprasolicitare indusă de operaţia de diagnosticare. Ulterior, când programul este compilat şi devine o versiune utilizabilă, aceste verificări sunt eliminate şi programul se execută la viteză maximă.

Există trei funcţii macro utilizate frecvent într-un program MFC:

- **ASSERT** prezintă o casetă de dialog cu un mesaj de eroare când primeşte o expresie evaluată drept falsă, această funcţie macro este compilată numai în variantele de depanare;
- **VERIFY** funcţionează exact ca ASSERT, cu deosebirea că expresia evaluată este întotdeauna compilată, chiar şi în variantele non-depanare, deşi expresia nu este testată în versiunile finale;
- **ASSERT_VALID** testează un pointer către un exemplar din clasa CObject şi verifică dacă obiectul este un pointer valid într-o stare validă. O clasă derivată din CObject poate invalida funcţia ASSERT_VALID pentru a permite testarea stării unui obiect.

Funcţiile macro ASSERT şi VERIFY sunt utilizate cu orice expresie, nu numai cu cele care implică CObject. Deşi ambele efectuează teste pentru a se asigura că expresia evaluată are valoarea **TRUE**, există o diferenţă importantă între modurile de operare ale acestora. Când se compilează pentru o versiune finală, funcţia ASSERT şi expresia evaluată de aceasta sunt complet ignorate în timpul compilării. Şi funcţia macro VERIFY este ignorată, dar expresia este compilată şi utilizată în versiunea finală.

O sursă comună de erori în programele MFC este *plasarea de coduri importante în interiorul unei funcţii macro ASSERT în loc de VERIFY*. Dacă acea expresie este necesară pentru ca programul să lucreze corect, trebuie amplasată în interiorul unei funcţii macro VERIFY şi nu ASSERT .

În afara funcţiilor de diagnosticare şi a celor macro, clasa CObject mai declară o **funcţie virtuală care afişează conţinutul unui obiect în timpul rulării**. Această funcţie dump, este utilizată pentru a transmite mesaje cu privire la starea curentă a unui obiect către o fereastră a programului de depanare.

Dacă se depanează un program MFC, mesajele sunt afişate într-o fereastră de ieşire a programului de depanare. Se va adăuga declaraţiei de clasă CObiectMeu codul sursă, funcţia dump este plasată, de obicei în secţiunea de implementare a declaraţiei de clasă. Deoarece dump este apelată numai de cadrul de lucru MFC, este de obicei declarată drept **protected** sau **private**. Deoarece funcţia dump este apelată numai în variantele de depanare, declaraţia este înconjurată de instrucţiunile #ifdef şi #endif, care elimină declaraţia funcţiei dump pentru versiunile finale.

Clasa CWnd este utilizată în mod normal ca un pointer generic de fereastră, şi conţine funcţii suplimentare ce pot fi aplicate majorităţii ferestrelor. Cele mai des folosite sunt:

- ✓ **ShowWindow**, este folosită pentru ascunderea sau afişarea unei ferestre; funcţia trebuie să aibă un parametru care să indice modul în care comanda influenţează fereastra (există 10 parametrii posibili cei mai frecvenţi sunt SW_HIDE pentru ascunderea ferestrei şi SW_SHOW, care afişează fereastra folosind dimensiunea şi poziţia curentă a acesteia);
- ✓ **SetWindowText** este utilizată pentru a stabili legenda sau bara de titlu a unei ferestre, dacă fereastra este un control de editare, se returnează conţinutului controlului;

- ✓ **GetWindowText** returnează legenda sau bara de titlu a unei ferestre;
- ✓ **MoveWindow** este utilizată pentru a preciza noua poziţie a unei ferestre sau a unui control;
- ✓ **SetFont** permite definirea corpului de literă utilizat pentru o fereastră, este valabilă pentru toate ferestrele, inclusiv pentru controalele de editare.
- ✓ **GetFont** returnează corpul de literă curent utilizat de o anumită fereastră.

□ *Clasa MFC CString*

Clasa **CString** reprezintă o implementare a obiectului şiruri tip matrice şi a metodelor aferente acestuia. Această clasă, creează şiruri inteligente care pot fi cu uşurinţă adunate, copiate sau manevrate în alt mod, fără a avea nici una dintre problemele care apar la lucrul cu matricele de caractere.

Unul dintre marile avantaje ale clasei CString este faptul că un **obiect CString** se comportă exact ca unul dintre tipurile fundamentale, ca un întreg. Un obiect CString poate fi adăugat, atribuit sau copiat exact ca un întreg, însă totuşi există anumite funcţii membru definite pentru clasa CString.

Clasa CString oferă o funcţie membru **GetLength** care returnează numărul de caractere stocate într-un obiect CString. Această funcţie este utilă la stocarea de obiecte CString sau la pregătirea pentru diverse tipuri de date de ieşire. Pentru localizarea în interiorul unui obiect CString a unui caracter sau a unui şir de caractere se poate folosi funcţia membru Find. Această funcţie este utilă atunci când se caută caracterele de control în şirurile de intrare.

Funcţia membru **SetAt** este utilizată pentru a transforma caracterul retur de car (CR) într-un spaţiu. Funcţia se poate folosi pentru a transforma orice caracter, atâta timp cât noul caracter nu este nul.

2.1.3.3. Clase tip colecţie

O clasă tip colecţie este utilizată pentru a crea obiecte capabile de a stoca alte obiecte, aşa cum este cazul claselor container. Obiectele stocate pot aparţine unuia dintre tipurile încorporate, cum ar fi int, char sau long, sau pot fi obiecte de tip CStrings. Există trei tipuri fundamentale de clase tip colecţie conţinute de biblioteca de clase MFC.

- **Matrice**. O colecţie tip matrice permite stocarea şi accesarea datelor utilizând operatorul indice sau [ ]. Clasa de matrice este optimizată pentru a permite un acces facil la un anumit articol din matrice, dacă poziţia sa este cunoscută. Totuşi inserarea unui articol în mijlocul unei matrice poate fi foarte costisitoare sub aspectul timpului de procesare.
- **Liste.** O colecţie tip listă îşi stochează articolele într-o listă tip lanţ (chain-like list), ceea ce facilitează în mare măsură adăugarea sau eliminarea articolelor, indiferent de amplasarea acestora. Colecţiile tip listă au un cap şi o coadă. Articolele pot fi inserate cu uşurinţă la capul sau la coada listei. Totuşi, căutarea unui articol într-o listă mare poate fi dificilă deoarece articolele din listă trebuie testate pentru a găsi un anumit articol; în medie, aceasta înseamnă testarea a aproximativ jumătate din articolele unei liste la fiecare căutare.
- **Hărţi**. O hartă este o colecţie ceva mai complexă. Fiecărui articol stocat într-o hartă îi este asociată o cheie unică.

Cheia fiecărui articol este unică şi este utilizată pentru accesul la articolul stocat în hartă. Adăugarea, eliminarea sau accesarea oricărui articol stocat într-o hartă este o operaţie foarte simplă; totuşi este necesară cheia articolului care trebuie recuperat. Această cheie este convertită într-un index folosit la accesarea fiecărui articol în parte. Fiecare cheie este convertită mai întâi într-o valoare codificată (hash value), care este folosită apoi ca indice într-o matrice cu articolele colecţiei, cunoscută sub numele de **tabel de codificare** (hash table). Dimensiunea acestui tabel se poate specifica la crearea hărţii; un tabel de mari dimensiuni impune spaţiu de stocare mărit, în timp ce un tabel redus măreşte posibilitatea ca două chei să genereze valori codificate egale, proces cunoscut sub numele de coliziune. Clasele MFC tip hartă tratează coliziunile prin crearea unor liste de articole care au chei egale.

Cele trei tipuri de colecţie sunt disponibile în versiuni şablon şi non şablon. Versiunile non-şablon, specifică un anumit tip de articol care urmează a fi stocat în colecţie.

Clasele tip colecţie MFC oferă o mare varietate de colecţii care se pot folosi în programe MFC. Clasele tip colecţie sunt furnizate ca parte a bibliotecii de clase, deci utilizarea acestora necesită foarte puţin efort de scriere din partea programatorului. Reutilizarea acestor clase permite economisirea a mii de linii de cod în aplicaţii.

Colecţia CList

Este folosită adeseori atunci când o colecţie de articole este utilizată ca un lanţ de articole înrudite, de exemplu dacă se doreşte accesarea articolelor de la începutul sau de la sfârşitul listei. Colecţia CList permite adăugarea de articole la începutul şi la sfârşitul listei. Pentru a insera un articol în capul unei liste din CList, se foloseşte funcţia membru *AddHead*.

Pentru a traversa sau a examina fiecare articol stocat într-o colecţie, este utilizată o variabilă **POSITION**. O variabilă **POSITION** este asemănătoare unui cursor într-un program de procesare a textelor. Prin apelarea funcţiei membru GetHeadPosition variabila **POSITION** este plasată exact anterior primului articol stocat în listă. Funcţia membru GetNext execută două operaţii: returnează următorul articol stocat pe listă şi avansează variabila **POSITION** către următorul articol din CList. Dacă această variabilă are valoarea **NULL**, atunci nu mai există articole în CList.

Toate clasele tip colecţie dispun de o funcţie RemoveAll, care este utilizată pentru a înlătura şi a şterge toate articolele stocate într-o colecţie.

Clasa tip colecţie CArray

Clasa CArray este utilizată pentru a crea colecţii care sunt mai utile decât matricele încorporate în **C++**. Clasa CArray asigură verificarea de domeniu, sub formă de apeluri ale funcţiei ASSERT la compilarea programului, cu activarea opţiunilor de depanare. De asemenea, permite dezvoltarea dinamică a matricei la adăugarea de elemente noi, aspect imposibil în cadrul matricelor încorporate.

Numărul de articole stocate în CArray poate fi furnizat prin apelarea funcţiei membru GetSize, accesul la diverse articole stocate în CArray poate fi realizat folosind operatorul de indexare (_), ca şi în cazul matricelor încorporate. Funcţia GetSize se foloseşte pentru garantarea faptului că articolul respectiv există, iar dacă nu există certitudinea stocării unui anumit membru, se va folosi în schimb funcţia GetAt.

Clasa tip colecţie CMap

Clasa CMap este folosită pentru crearea colecţiilor care se indexează folosind o cheie unică pentru fiecare articol stocat în colecţie. Fiecare articol stocat în colecţie trebuie să aibă o cheie unică; dacă se încearcă stocarea a două articole cu aceiaşi cheie, primul articol va fi înlocuit de către al doilea articol.

Clasa tip colecţie CMap este ideală pentru stocarea articolelor care deţin numere de identitate sau de serie unice. De exemplu, mărcile salariaţilor, seria împreună cu numărul buletinului de identitate sau indicii ISBN ai cărţilor reprezintă toate exemple bune de chei, deoarece sunt în mod garantat unice.

Lista de parametri ai şablonului CMap este mult mai complicată decât listele de parametrii folosite pentru CList şi CArray. Primii doi parametrii sunt folosiţi pentru tipul de cheie. Primul parametru reprezintă tipul de cheie folosit ca argument în funcţiile membru.

Următorii doi parametrii se referă la articolul stocat în colecţie. Al treilea parametru reprezintă tipul de articol stocat în colecţie, iar al patrulea parametru reprezintă tipul de articol folosit ca argument în funcţiile membru.

Funcţia membru SetAt este folosită pentru stocarea unui articol şi a unei chei în colecţia CMap. Funcţia membru CMap este adesea folosită pentru recuperarea unui articol din colecţie, în funcţie de o anumită cheie. În colecţiile de mari dimensiuni, CMap este cu mult mai rapidă decât CList sau CArray în materie de căutare, inserare sau ştergere a unui articol arbitrar, în

funcţie de o cheie.

Căutarea într-o CMap este foarte asemănătoare cu procesul analog într-o CList. Funcţia membru GetStartPosition returnează o variabilă **POSITION** care este avansată către articolul următor folosind funcţia membru GetNextAssoc. Totuşi, în afară de valoarea cheii, nu există nici o ordine a articolelor stocate într-o CMap. Dacă CMap este traversată folosind funcţia GetNextAssoc, articolele din colecţie sunt returnate în funcţie de structura internă a CMap, şi nu în funcţie de ordinea în care au fost inserate.

UTILIZAREA BAZELOR DE DATE, PROGRAMAREA OLE ŞI COM

2.2.1. Utilizarea bazelor de date

Organizaţiile cu profil economic acumulează o cantitate mare de informaţii în urma activităţilor comerciale de zi cu zi. Sunt înregistrate detalii despre clienţi şi furnizori, despre vânzări şi necesarul de stocuri etc. Marea majoritate a corporaţiilor optează pentru stocarea acestor date vitale în cadrul bazelor de date sau, mai exact, într-un **DMS** (**DataManagementSystem - sistem de gestionare a bazelor de date**). Un sistem de gestionare a bazelor de date este un produs software care stochează datele într-o structură bine organizată şi oferă mijloace eficiente de accesare şi actualizare a acestor date.

Există două tipuri de baze de date: **baze de date relaţionale** şi **baze de date obiectuale**. Diferenţa dintre ele provine din modul conceptual în care sunt gestionate datele.

Bazele de date relaţionale *au la bază tipuri de date simple, recunoscute* (caractere, şiruri, întregi etc.) *şi nu permit crearea unor noi tipuri de date.*

Bazele de date obiectuale *operează cu tipurile de date la un nivel mai înalt, permiţând crearea prin definire a unor noi tipuri de date*. Aceste obiecte corespund celor create într-un limbaj orientat-obiect, de exemplu un obiect stocat într-o bază de date obiectuală ar putea fi o persoană sau un automobil.

Există mai mulţi producători care oferă baze de date relaţionale, fiecare bază de date are propria sa structură şi, în consecinţă, un set propriu de funcţii API. Din ce în ce mai mult se cere ca aplicaţiile să poată lucra cu orice bază de date. Pentru a putea dezvolta aplicaţii care să utilizeze o bază de date relaţională, indiferent de producătorul acesteia, este nevoie de o metodă generică de programare. O astfel de metodă există şi se numeşte **ODBC (Open Database Connectivity - conectivitate deschisă a bazelor de date).**

ODBC aduce o interfaţă de programare standard prin care pot fi utilizate diferite tipuri de baze de date relaţionale. În acest scop este necesară existenţa unui intermediar care să traducă apelurile ODBC standard în apeluri de funcţii specifice bazei de date. Acest intermediar este driverul ODBC, oferit de către producătorul bazei de date sau de către o firmă terţă specializată în astfel de produse. ODBC a fost adoptat ca standard şi astfel de drivere există acum pentru toate bazele de date răspândite.

Procedura de instalare a unor aplicaţii Microsoft, precum Office şi Visual Studio, permite instalarea celor mai folosite drivere ODBC produse de Microsoft.

Comenzile sunt transmise driverului ODBC şi pasate mai departe bazei de date cu ajutorul SQL (limbaj structurat de interogare). Acest limbaj a fost dezvoltat anume pentru accesarea bazelor de date şi este acum standardul de facto. Există aproape tot atâtea variante de SQL câte baze de date sunt, fiecare producător aducând propriile îmbunătăţiri. Dar există cerinţe minime care asigură un set de comenzi care nu sunt standard, deoarece ODBC oferă o metodă de scurt-circuitare ce permite execuţia directă a instrucţiunilor SQL. Utilizarea unor comenzi în afara standardului înseamnă ca aplicaţia să-şi piardă capacitatea de a accesa bazele de date ale altor producători, tocmai această capacitate fiind avantajul major adus de ODBC.

Limbajul structurat de interogare **SQL** conţine trei tipuri de comenzi: DDL, DML şi DCL.

Comenzile DDL (Data Definition Language - limbaj pentru definirea datelor) sunt folosite *pentru crearea şi modificarea schemelor de baze de date.*

Comenzile DML (Data Manipulation Language - limbaj pentru manipularea datelor) sunt folosite *pentru interogarea şi manipularea informaţiilor*.

Comenzile DCL (Data Control Language - limbaj pentru controlul datelor) sunt folosite pentru *acordarea unor drepturi de acces specifice diferiţilor utilizatori*.

Prima sarcină legată de utilizarea ODBC este configurarea unei surse de date. O sursă de date indică programului unde se află fişierele care conţin baza de date şi ce driver ODBC trebuie folosit pentru interpretarea administratorului ODBC pentru sursele de date, accesibil din panoul de control.

2.2.2. Noţiuni de programare OLE şi COM

Complexitatea crescândă a dezvoltării aplicaţiilor în cadrul unor medii complexe, aşa cum este Windows, cu multele sale interfeţe de programare a aplicaţiilor (API), precum şi nevoia de standardizare, de control al versiunilor, de accelerare a dezvoltării şi de calcul distribuit, au dus la apariţia unei tehnologii numite **programare bazată pe componente**.

Următorul pas în evoluţia COM va fi COM +, Microsoft promite că acesta va simplifica în bună măsură scrierea codului necesar pentru COM, astfel că obiectele COM vor avea un comportament mai similar cu obiectele **C++** şi vor putea fi create prin intermediul cuvintelor cheie **new** şi **delete** din **C++.**

Componentele sunt entităţi software de dimensiuni mici (asemeni instanţelor unei clase) care efectuează operaţii specifice prin intermediul unor interfeţe bine definite. Spre deosebire de instanţele claselor, componentele nu sunt legate definitoriu de o instanţă a unui program sau de un sistem gazdă, pot fi scrise într-o multitudine de limbaje, şi cu toate acestea, comunică fără probleme, prin intermediul interfeţelor, cu programe şi componente implementate în alte limbaje.

Microsoft a dezvoltat această tehnologie până la forma actuală, fiind denumită în prezent **Component Object Model (COM)** - modelul componentelor obiectuale. Se pot întâlni şi alţi temeni înrudiţi, precum **legarea şi încapsularea obiectelor (Object Linking and Embedding-OLE)** sau **controale ActiveX**. Acestea reprezintă implementări particulare ale programării COM. Propriu-zis COM este un standard independent de limbaj şi de platformă care defineşte modul în care obiectele pot comunica între ele prin intermediul unui protocol acceptat în comun.

Cel mai important element privind obiectele COM îl reprezintă **interfeţele** acestora; obiectele propriu-zise nu sunt decât cutii negre care implementează o funcţionalitate particulară. Pentru a putea să se utilizeze această funcţionalitate, programele trebuie să respecte un contract bine definit privind transmiterea parametrilor şi obţinerea rezultatelor. Acest contract dintre programul client şi obiect se numeşte **interfaţă.**

Întregi biblioteci **API** pot fi definite şi proiectate în termeni de interfaţă. Dacă se dezvoltă o aplicaţie client, se poate scrie programul astfel încât să comunice cu un server prin intermediul acestor interfeţe acceptate, fără a mai conta ce aplicaţie server va fi utilizată. Reciproc, se poate decide dezvoltarea unor componente de tip server care respectă definiţiile interfeţelor, comercializându-le ca alternativă viabilă la componentele altor producători.

Interfaţa de programare a aplicaţiilor pentru mesagerie reprezintă un set de interfeţe standard pentru obiectele COM. Oricine este liber să scrie obiecte COM care efectuează operaţii precum *stocarea mesajelor, transportul mesajelor şi oferirea către destinatarii mesajelor a unei agende de adrese*.

Microsoft Exchange este o implementare particulară a acestor componente server, dar există multe alte implementări. Codul efectiv s-ar putea să difere enorm între diferite implementări, dar toate obiectele **COM** respectă aceleaşi specificaţii de interfaţă. Aceasta înseamnă că, toţi clienţii acestor servicii, Microsoft Outlook, pot folosi componentele compatibile **MAPI** ale oricărui producător în scopul trimiterii, primirii şi stocării mesajelor. În

mod similar, orice producător poate să ofere propriile programe client (şi mulţi o fac) care folosesc Exchange sau orice alte componente server MAPI, chiar fără a şti ale cui sunt componentele utilizate. Singura cerinţă în ceea ce priveşte componentele client şi server este ca acestea să se apeleze reciproc prin intermediul acestor interfeţe definite de comun acord, reunite sub numele de MAPI.

Interfeţe COM

O interfaţă este o definiţie a unei mulţimi de funcţii şi a parametrilor acestora. Orice obiect COM dispune de cel puţin o interfaţă, iar în mod frecvent sunt oferite mai multe interfeţe, fiecare conţinând propriul său set unic de funcţii.

Un obiect COM poate fi scris în orice limbaj care are capacitatea de a suporta aceste interfeţe. Unele limbaje sunt mai potrivite în acest sens decât altele, de exemplu JAVA este foarte potrivit, din cauză că, fiecare obiect JAVA poate avea mai multe interfeţe, deci se mapează natural peste un obiect COM. Obiectele COM conţin codul efectiv aflat în spatele acestor interfeţe, astfel că un program care apelează o funcţie declarată într-o interfaţă va găsi o implementare ce efectuează operaţia definită de acea funcţie în cadrul interfeţei respective.

Structura interfeţei COM standard este exact aceeaşi cu structura tabelei de funcţii virtuale din Visual C++, ceea ce înseamnă că este posibil să se folosească mecanismul tabelelor de funcţii virtuale pentru a defini şi implementa interfeţe COM.

În mod normal, funcţiile virtuale sunt folosite ca o metodă de supradefinire în cadrul unei clase derivare a unei funcţii din clasa de bază. În acest scop este declarată o tabelă de funcţii virtuale asociată clasei respective.

Orice instanţă de memorie a unui obiect C++ are ataşată o tabelă de funcţii virtuale (chiar dacă se poate ca unele tabele să nu conţină nici o intrare şi deci, să fie vide). Fiecare intrare din tabelă conţine un pointer către codul care implementează o funcţie virtuală. De fiecare dată când se apelează una dintre funcţiile virtuale ale obiectului, tabela ataşată oferă adresa corectă, corespunzătoare fie funcţiei din clasa de bază, fie funcţiei din clasa derivată.

Declararea unei clase C++ care conţine exclusiv funcţii virtuale pure, se numeşte **clasă de bază abstractă**. Nu este posibilă instanţierea unei clase abstracte, dar aceste clase se pot utiliza pentru crearea unei definiţii de interfaţă COM compatibilă cu C++. La crearea unei instanţe a unui obiect COM, un proces server aflat în rulare se va ocupa de iniţializarea şi gestionarea acesteia. Procesul server poate fi propriul program client, un DLL ataşat, un executabil (.exe) aflat pe propriul calculator sau un program aflat pe un sistem la distanţă. Fiecare din aceste ipostaze diferite se numeşte **context**.

Când se apelează funcţii pe baza acestui pointer la interfaţă, propriul client şi obiectul COM sunt puse în legătură cu ajutorul unui DLL terţ, care se numeşte **de intermediere**, tehnica purtând numele de **apel intermediat.** Biblioteca de intermediere are în sarcină *împachetarea parametrilor într-un format independent de maşină, în scopul transmisiei*. Este posibil apoi, să folosească apelurile de procedură la distanţă pentru a apela o funcţie a unui obiect COM aflat la distanţă. Bibliotecile de intermediere pot fi generate automat, prin crearea definiţiilor în limbajul de definire a interfeţelor (IDL) şi prelucrarea acestora de către compilatorul Microsoft IDL.

Vechea problemă a întreţinerii versiunii corecte a aplicaţiei este rezolvată printr-o simplă regulă. Odată ce s-a eliberat un obiect COM pentru a fi folosit de utilizatorii de pretutindeni, versiunile ulterioare trebuie să păstreze ne-alterate interfeţele originale. Aceasta înseamnă că noile versiuni ale unui obiect COM trebuie să implementeze interfeţe suplimentare, fără a le modifica pe cele originale. Programele client mai vechi, care nu ştiu să utilizeze decât interfaţa mai veche, vor solicita în continuare această interfaţă, în timp ce programele client care cunosc noile facilităţi pot să solicite versiunea mai nouă de interfaţă.

Automatizarea OLE

Legarea şi încapsularea obiectelor (OLE) este un termen folosit iniţial pentru a *descrie capacitatea de a insera obiecte de diferite tipuri în documente având un tip propriu*, un exemplu

fiind inserarea foilor de calcul Excel în documentele Word.

Documentele care suportă operaţiile de legare şi încapsulare se numesc **documente compuse**. Obiectele inserate pot fi încapsulate, ceea ce înseamnă că pot fi salvate odată cu documentul, sau legate, ceea ce înseamnă că în document va fi inserată o referinţă la alt fişier (numită identificator). Acestea sunt cele două operaţii care au dat naştere termenului OLE original.

Semnificaţia acestui termen, s-a lărgit însă, incluzând acum **tragerea şi plasarea** OLE şi **automatizarea** OLE, acestea din urmă referindu-se la situaţia în care *un program poate să apeleze metodele unui alt program care rulează ascuns, fără ca utilizatorul să fie conştient de această interacţiune*.

Un obiect COM care oferă o interfaţă dispecer conţine o tabelă de metode (funcţii), evenimente (funcţii care utilizează apeluri inverse ale clientului în scopul unei notificări a acestuia) şi proprietăţi (funcţii care furnizează sau stabilesc valoarea unei anumite variabile a obiectului COM). Programele client pot să acceseze aceste informaţii dinamic (procedeul se numeşte legare târzie), în timpul rulării, pentru a afla detalii privind obiectul de automatizare.

Se poate întâlni termenul de interfaţă dual folosit în legătură cu COM şi OLE. Un obiect COM poate să implementeze o interfaţă dual, oferind funcţiile atât printr-o interfaţă directă COM, cât şi printr-o interfaţă dispecer. Astfel, programele client pot beneficia de avantajele ambelor soluţii; programele C++ rapide pot să acceseze obiectul direct prin interfaţa COM rapidă, iar Visual Basic şi limbajele de scriptare pot să acceseze funcţiile prin interfaţa dispecer mai înceată.

- Crearea unui server propriu de automatizare OLE

Multe aplicaţii, printre care Word, Excel sau chiar Developer Studio, sunt **servere de automatizare**. Acestea pun la dispoziţia aplicaţiilor client o interfaţă dispecer care poate fi utilizată pentru apelarea funcţiilor proprii, oferind servicii specializate precum verificarea ortografică.

Visual Basic Scripting, instrumentul utilizat pentru crearea macro-comenzilor, foloseşte aceste metode în scopul creării şi manipulării documentelor din cadrul serverelor de automatizare. De fiecare dată când se scrie o macro-comandă pentru unul din aceste programe, se foloseşte o versiune redusă de Visual Basic care permite apelarea funcţiilor din interfaţa dispecer a serverului de automatizare respectiv.

Serverele de automatizare sunt asociate de obicei cu un nume inteligibil care poate fi utilizat pentru determinarea identificatorilor de clasă corespunzători, care sunt reţinute într-un registru intern, conţinând subchei ce specifică identificatorul de clasă al serverului de automatizare respectiv.

Infrastructura unei aplicaţii server de automatizare poate fi creată prin intermediul AppWizard, validând opţiunea **Automation** din pasul trei din MFC AppWizard.

- Servere şi containere OLE

Un **server** OLE este *o aplicaţie care poate fi lansată în cadrul unei ferestre a unei aplicaţii container*. De exemplu, foile de calcul Excel, pot fi inserate şi editate în Word. În acest caz, Word reprezintă aplicaţia container OLE, iar Excel este aplicaţia server OLE.

Se mai întâlneşte termenul de **document activ**, folosit în legătură cu serverele şi containerele OLE, ceea ce este o terminologie relativ nouă, legate de ActiveX, care dezvoltă **arhitectura container/server** OLE pentru a permite obiectelor încapsulate să câştige controlul asupra întregii zone client a containerului (nu doar asupra unui mic cadru) şi să manipuleze direct fereastra, meniurile şi barele cu instrumente ale acestuia.

Aplicaţiile pot fi concomitent containere OLE şi servere OLE, caz în care pot juca ambele roluri. Word-ul şi Excel-ul sunt exemple potrivite în acest sens, deoarece se pot rula Excel şi să se insereze documente Word şi viceversa.

Un server complet poate să ruleze ca aplicaţie de sine stătătoare sau ca obiect inserat, în

timp ce mini-serverele pot fi lansate doar din cadrul unui program container. La inserarea unui obiect server OLE, acesta poate fi editat în interiorul unei suprafeţe rectangulare care este mărginită de un chenar gros, haşurat, de culoare neagră, numit cadru local. Operaţia se numeşte **activare locală** şi este iniţiată prin transmiterea unor indicatori, numiţi verbe, de la container către server. Atunci când nu este activ, cadrul dispare şi imaginea afişată este ultima generată atunci când acesta era activ (stocată şi reprodusă pe baza unui context dispozitiv de tip metafişier). Atunci când este activ, cadrul este afişat, iar la meniul propriu al container-ului sunt adăugate elemente de meniu ale obiectului server.

Datele documentului încapsulat pot fi serializate de către documentul container prin intermediul funcţiilor obişnuite de serializare. Dacă un document încapsulat este stocat integral în cadrul documentului compus al aplicaţiei client, obiectele legate sunt stocate sub forma unor simpli identificatori. Un astfel de identificator este un mic obiect care identifică în mod unic locaţia datelor propriu-zise şi modul de afişare a acestora în cadrul aplicaţiei client.

Infrastructura standard pentru **servere** OLE şi **containere** OLE poate fi creată cu ajutorul AppWizard, însă cu toate acestea, între container şi server există o cooperare complexă care trebuie implementată pentru a asigura funcţionarea corespunzătoare a ambelor părţi. Prin personalizarea a două clase adăugate pe partea de server şi a clasei suplimentare a containerului se poate implementa suport pentru operaţii destul de complexe de editare locală la rularea într-un cadru în fereastra unei aplicaţii container. Nici serverul şi nici containerul nu au voie să cunoască tipul obiectului care încapsulează sau al celui conţinut. Atât timp cât obiectele server şi client respectă acelaşi standard, containerul şi serverul pot fi integrate perfect pentru a da utilizatorului impresia unei aplicaţii unitare.

2.3. TIPURI SI STRUCTURI DE DATE. TIPURI DE DATE ABSTRACTE

Studiul datelor cuprinde următoarele probleme:

- maşini pentru stocarea datelor
- limbaje de descriere a manipularii datelor
- fundamente care descriu ce date putem obţine prelucrând date primare
- structuri de reprezentare a datelor

Obiectivele studiului structurilor de date sunt următoarele:

- explorarea diverselor modalităţi de structurare a datelor
- identificarea clasei de operaţii care pot executate asupra fiecărei structuri
- modul de reprezentare în calculator al obiectelor aparţinând structurii respective astfel incât aceste operaţii să poată fi executate cât mai eficient

Scopul studiului structurilor de date este reprezentat de însuşirea şi stăpânirea următoarelor două tehnici fundamentale:

- proiectarea unor reprezentări adecvate ale datelor
- proiectarea şi analiza algoritmilor care operează cu aceste reprezentări

Orice constantă, variabilă, expresie sau funcţie se caracterizează printr-un anumit tip de date. Prin tip de date se înţelege o multime de elemente numite valori sau obiecte, care pot clar distinse intre ele. Într-un limbaj de programare, orice constantă aparţine unui anumit tip de date, orice variabilă poate stoca valori de un anumit tip, orice expresie se evaluează la o valoare de un anumit tip şi orice funcţie întoarce o valoare de un anumit tip.

După modul de definire în raport cu un anumit limbaj de programare, tipurile de date se clasifică în:

- tipuri predefinite - limbajul permite folosirea variabilelor de tipul respectiv şi furnizează o mulţime de operaţii predefinite pentru manipularea acestor variabile în cadrul unui program.
- tipuri definite de utilizator

Obiectele unui limbaj de programare sunt:

- constante: tipul rezultă din forma sintactică (modul de scriere) a lor
- variabile: tipul rezultă prin specificarea în program a unui enunţ special – declaraţie
- expresii: tipul rezultă din tipurile operanzilor şi modul lor de compunere cu ajutorul operatorilor.
- funcţii: tipul rezultă din declaraţia funcţiei (în C - prototip sau semnătură).

Declaraţia este un enunţ care asociază o variabilă sau o funcţie cu tipul său. Definiţia este un enunţ care asociază o variabilă sau funcţie cu tipul său şi alocă spaţiu de memorie variabilei sau corpului funcţiei respective.

Dupa modul de definire în termenii altor tipuri de date, tipurile de date se clasifică în:

- tipuri primitive
- tipuri derivate

Un obiect poate fi:

- compus: se poate descompune în mai multe componente
- simplu sau scalar.

La rândul său, un tip de date poate fi:

- compus: valorile sale sunt compuse
- simplu sau scalar

Un tip de date este ordonat dacă pe multimea valorilor sale se poate defini o relatie de ordine. În caz contrar, se spune că este neordonat. Un tip de date este ordinal dacă pe mulţimea elementelor sale se poate defini o relaţie de ordonare liniară (este posibilă definirea predecesorului şi succesorului unui element). Un tip ordinal modelează o mulţime finită sau numărabilă.

Cardinalitatea unui tip T reprezintă numărul de valori distincte aparţinând tipului T.

O structură de date se defineşte ca mulţimea de obiecte care alcătuiesc un anumit tip, proprietăţile obiectelor, relaţiile dintre obiecte şi operaţiile asupra obiectelor. Definiţia unei structuri de date trebuie să conţină trei elemente:

i) numele structurii
ii) operaţiile asupra structurii
iii) proprietăţile structurii

Operaţiile de bază aplicabile obiectelor unei structuri de date sunt:

- constructori – se referă la modul de generare a noi obiecte apartinand structurii respective;
- destructori – se utilizează atunci când gestiunea obiectelor se face explicit de către programator; eliberează memoria alocată obiectelor;
- selectori - se referă la modul de accesare a elementelor care compun un obiect compus;
- modificatori – se referă la modul de actualizare a elementelor care compun un obiect compus;
- recunoscători – se utilizează pentru recunoaşterea diverselor proprietăţi ale obiectelor aparţinând structurii respective;
- testori – se utilizează pentru compararea obiectelor aparţinând structurii respective

Proprietăţile structurii nu descriu modul în care trebuie implementate operaţiile structurii. Cu toate acestea, proprietăţile unei structuri de date exprimă maniera în care trebuie să funcţioneze operaţiile structurii, indiferent de modul lor de implementare. Cu alte cuvinte, proprietăţile abstractizează operaţiile de modul lor de implementare. Din acest motiv, noţiunea de structură de date este cunoscută şi sub numele de **tip abstract de date,** iar această manieră de definire a structurilor de date se numeşte abstractizarea datelor.

Formal, o structură de date poate fi definită ca un tuplu S = {D, d, O, P}, unde:

i) D este o mulţime de tipuri de date implicate în definirea lui S
ii) d € D este tipul de date al obiectelor structurii S
iii) O este mulţimea operaţiilor definite pe structura S
iv) P este mulţimea proprietăţilor structurii S

O implementare a unei structuri de date S este o transfomare care defineşte modul de reprezentare a obiectelor aparţinând tipului de date al structurii în funcţie de obiectele altei structuri de date T. Fiecare operaţie a lui S trebuie definită în termenii operaţiilor lui T

Cele mai întâlnite tipuri primitive în limbajele moderne de programare sunt:

a) **tipuri numerice**: ele sunt reprezentări ale unor mulţimi de numere cunoscute din matematică, precum N, Z sau R. Vom avea în consecinţă date întregi fără semn, întregi cu semn sau reali. Numerele întregi se reprezintă intern în calculator sub forma unei succesiuni de biţi, cu ajutorul codului complement (cod complement faţă de 2). Numerele reale se reprezintă intern în calculator cu ajutorul reprezentării în virgulă mobilă (flotantă), care cuprinde semnul, mantisa şi exponentul. Pentru a reprezenta corect şi numerele subunitare, se utilizează reprezentarea cu semn, caracteristică şi exponent.

b) **tipul caracter**: este destinat reprezentării caracterelor alfanumerice. Tipul caracter modelează mulţimea finităa a tuturor caracterelor afişabile. Din nefericire, nu există un set standard de caractere specific tuturor sistemelor de calcul. Din acest motiv, termenul "standard" trebuie interpretat în acest context ca referindu-se la sistemul de calcul particular pe care este executat programul în cauză. Cel mai folosit set "standard" de caractere este cel definit de Organizaţia Internaţională pentru Standarde (ISO) şi anume versiunea sa americană (codul ASCII), care constă din 128 de caractere, dintre care primele 33 sunt caractere de control şi celelalte 95 sunt caractere afişabile.Caracterele sunt codificate intern prin valori întregi numite coduri. Deoarece codul ASCII are 128 de caractere, pentru codificarea caracterelor sale sunt suficienţi 7 biţi. Cu toate acestea se foloseşte pentru codifiare un octet, deoarece memoria internă a sistemelor de calcul este structurată de obicei sub forma unei secvenţe de locaţii cu dimensiunea de un octet.Pentru seturile cu peste 256 de caractere, a fost introdusă specificaţia standard Unicode (cum este cazul limbii chineze sau japonenze), pentru reprezentarea căreia se folosesc doi octeţi.

c) **tipul logic**: este destinat reprezentării mulţimii valorilor logice fals şi adevărat.

d) **tipul enumerare:** este destinat reprezentării unei mulţimi finite de valori, fiecare valoare fiind desemnată printr-un nume simbolic. Se utilizează de obicei atunci când este necesară definirea unui tip de date prin indicarea explicită a elementelor sale. Elementele unui tip enumerare se reprezintă intern prin codificare cu ajutorul unor constante întregi.

e) **tipul subdomeniu**: este destinat reprezentării unui interval al unui tip ordonat liniar. Tipurile subdomeniu se utilizează de obicei pentru reprezentarea indicilor tablourilor.

f) **tipul referinţă:** este destinat reprezentării adreselor altor obiecte. Tipul referinţă se utilizează de obicei pentru implementarea unor obiecte abstracte complexe cum sunt listele şi arborii. Numeroase prelucrări se referă la relaţiile dintre obiectele prelucrate şi nu doar la valorile lor. În astfel de cazuri poate fi necesară reprezentarea explicită a acestor relaţii. Pentru aceasta anumite obiecte trebuie să se poată referi la alte obiecte. Este posibil chiar ca un acelaşi obiect să fie referit din două sau mai multe locuri. Întrucât copierea valorii obiectului în locul de unde a fost referit nu este o soluţie acceptabil din cauza consumului mare de memorie şi dificultăţilor de menţinere a consistenţei, rezultă că trebuie apelat la o altă tehnică şi anume de a defini în program obiecte care să poată referi alte obiecte. Pentru aceasta se folosesc tipul referinţă şi obiectele pointer. O referinţă la un obiect se implementează ca adresă a locaţiei de memorie, corespunzătoare obiectului respectiv.

Fiind dat un obiect o de tip T , el poate fi referit utilizând un obiect de tipul referinţă la T . Vom nota acest tip prin Ref(T), definiţia sa axiomatică fiind ilustrată în figura următoare:

structura $Ref(\mathcal{T})$
operaţii
$vid :\rightarrow Ref(\mathcal{T})$
$ref : \mathcal{T} \rightarrow Ref(\mathcal{T})$
$deref : \{X \in Ref(\mathcal{T}) : X \neq vid\} \rightarrow \mathcal{T}$
proprietăţi
$(ref(X) = vid) = fals$
$ref(deref(P)) = P$
$deref(ref(X)) = X$

Constanta *vid* desemnează valoarea unui pointer care nu se referă la nici un alt obiect (o referinţă vidă).

Tipul referinţă la T se reprezintă în limbaj pseudocod prin *ref (T)* , ca în exemplul următor:

var int x, y
var ref int px
x ← 10
px ← ref(x)
y ←deref(px)

În urma executării acestei secvenţe de instrucţiuni, variabila *y* primeşte aceeaşi valoare ca şi *x*, adică 10, deoarece *px* are ca valoare o referinţă la *x*.

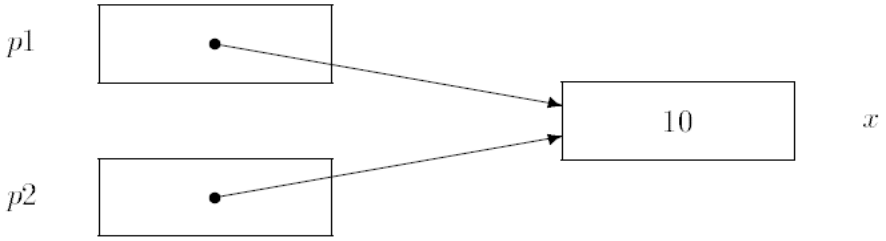
Un alt exemplu care realizează în mod diferit aceeaşi prelucrare este următorul:

var int x, y
var ref int px, py
x ← 10
px ← ref(x)
py ← ref(y)
deref(px) ← deref(py)

Se obişnuieşte ca "legăturile" create între obiecte prin intermediul pointerilor să se reprezinte grafic. Spre exemplu, efectul executării secvenţei:

var int x
var ref int p1, p2
x ← 10
p1 ← ref(x)
p2 ← ref(x)

se poate reprezenta grafic astfel:



O referinţă la un obiect se implementează ca adresă a locaţiei de memorie, corespunzătoare obiectului respectiv. Operaţia *ref* devine astfel operaţia de încărcare a adresei unui obiect. Operaţia *deref* se implementează utilizând tehnica adresării indirecte, prin care o anumită valoare este interpretată drept adresa unei locaţii de memorie de unde se va extrage valoarea propriu-zisă.

Alocare statică şi alocare dinamică

În cele discutate până acum, singura modalitate de a crea obiecte a constituit-o declararea acestora. O declaraţie are ca efect alocarea memoriei pentru obiectul respectiv la momentul compilăarii programului, obiectul fiind creat la încărcarea programului, deci înaintea execuţiei sale efective. Această modalitate de alocare a memoriei se numeşte statică. Principala caracteristică a acestui mod de alocare este faptul că obiectele sunt referite în program prin identificatorul introdus la declararea lor. Acest lucru este posibil deoarece asocierea dintre identificator şi locaţia de memorie rezervată obiectului s-a făcut în faza de compilare a programului.

Obiectele pot fi însă create şi la momentul execuţiei. Această modalitate de alocare a memoriei se numeşte alocare dinamică. Un astfel de obiect nu va putea fi însă accesibil prin program utilizând un identificator, deoarce momentul execuţiei succede momentului compilării programului. Din acest motiv, un obiect creat dinamic se numeşte şi obiect anonim.

O modalitate elegantă de a accesa obiectele create dinamic o constituie utilizarea tipului referinţă. Pentru aceasta se introduce operatorul *nou* care primeşte un tip T şi creează un obiect aparţinând lui T , întorcând o referinţă la T. Fie τ mulţimea tuturor tipurilor şi Ref mulţimea tuturor referinţelor. Semnătura acestui operator este *nou* : $\tau \rightarrow$ Ref .

Dacă operatorul nou întoarce referinţa vidă *vid*, atunci înseamnă că obiectul nu a putut fi creat. Dacă valoarea întoarsă este $p \neq vid$, atunci obiectul a fost creat şi el poate fi accesat utilizând "numele" *deref(p)*.

Operaţia inversă alocării dinamice a unui obiect este eliberarea memoriei alocate. Această operaţie se poate realiza automat, atunci când obiectul respectiv nu mai este referit în program, sau manual de către programator prin apelul operatorului standard *elib : Ref* →. Un exemplu de folosire este următorul:

```
var int x, y
var ref int p
...
p  ← nou(int)
deref(p) ← x
x ← y
y ← deref(p)
elib(p)
```

Tipuri de date compuse

Din punct de vedere conceptual, orice obiect compus este un n-tuplu $o = (o_1, o_2, ..., o_n)$, de obiecte o_i, $1 \leq i \leq n$, fiecare obiect o_i având tipul Ti. Tipul lui o este tipul compus T = (T1, ..., Tn). Fiecare dintre tipurile Ti poate fi la rândul său primitiv sau compus.

Pentru orice tip compus, pe lângă operaţiile fundamentale de atribuire şi test de egalitate/inegalitate, se mai definesc de obicei următoarele două:

- operatorul constructor, care construieşte un obiect compus $o = (o_1, o_2, ..., o_n)$, din obiectele componente o_i

- operatorul de selecţie, care permite accesul atât în citire cât şi în scriere la fiecare dintre componentele unui obiect compus

Cele mai des utilizate tipuri de date compuse predefinite de majoritatea limbajelor de programare sunt:

- **tipul agregat**

Un obiect agregat, numit şi înregistrare sau structură, este un obiect compus ale cărui componente (numite şi câmpuri) sunt eterogene (au tipuri diferite) şi au fiecare asociat câte un nume simbolic, folosit în cadrul operaţiei de selecţie.

Cardinalitatea tipului compus agregat este egală cu produsul cardinalităţilor tipurilor sale componente. La nivelul implementării, unui obiect agregat îi corespunde o zonă compactă de memorie a cărei mărime rezultă din mărimile componentelor obiectului repectiv – suma acestora.

Orice referinţă la un câmp al unui obiect agregat se transformă în distanţa zonei alocate acestuia faţă de începutul zonei alocate obiectului. Distanţele se calculează de regulă la momentul compilării programului. Accesarea unui câmp al un agregat se implementează de regulă utilizând tehnica adresării relative.

structura $Agregat(\mathcal{S}, \tau, \mathcal{T})$
operaţii
$(\ldots) : \times_{I \in \tau} T \rightarrow Agregat(\mathcal{S}, \tau, \mathcal{T})$
$sei(S) : Agregat(\mathcal{S}, \tau, \mathcal{T}) \times \mathcal{T}(S) \rightarrow Agregat(\mathcal{S}, \tau, \mathcal{T})$
$get(S) : Agregat(\mathcal{S}, \tau, \mathcal{T}) \rightarrow T(S)$
proprietăţi
$get(S_1)(sei(S_2)(A, T)) = (S_1 = S_2 : T\ ?\ get(S_1)(A))$

Reprezentarea axiomatică a tipului Agregat

- **tipul tablou**

Deseori este necesară folosirea unui tip compus ale cărui componente sunt toate de acelaşi tip. Un astfel de tip se numeşte omogen. În plus, accesul la componente are loc prin selectori care sunt elemente ale unei mulţimi finite şi ordonate liniar. În aceste situaţii se va folosi tipul tablou.

Tipul tablou se defineşte pornind de la un tip de bază T care modelează tipul componentelor şi de la un tip subdomeniu I, care modelează mulţimea selectorilor, numiţi în acest caz indecşi. Cardinalitatea tipului tablou se determină ridicând cardinalitatea tipului de bază la puterea cardinalităţii tipului indecşilor.

structura $Tablou(\mathcal{T}, \mathcal{I})$
operaţii
$(\ldots) : \times_{I \in \mathcal{I}} T \rightarrow Tablou(\mathcal{T}, \mathcal{I})$
$set : Tablou(\mathcal{T}, \mathcal{I}) \times \mathcal{I} \times \mathcal{T} \rightarrow Tablou(\mathcal{T}, \mathcal{I})$
$get : Tablou(\mathcal{T}, \mathcal{I}) \times \mathcal{I} \rightarrow \mathcal{T}$
proprietăţi
$get(set(T, I, X), J) = (I = J\ :\ X\ ?\ get(J, T))$

Reprezentarea axiomatică a tipului Tablou

La nivelul implementării, unui tablou îi corespunde o zonă contiguă de memorie, caracterizată prin adresă de început şi mărime. Să presupunem că min şi respectiv max sunt limitele inferioară şi respectiv superioară ale indicilor tabloului T şi să presupunem de aasemenea că pentru reprezentarea unui element al tabloului, aparţinând tipului de bază T , sunt necesare n cuvinte de memorie. Rezultă că zona de memorie alocată tabloului va avea (max-min+1) x n celule, iar zona în care este memorat elementul T[I] va începe la adresa A = B + (I - min) x n, unde B este adresa de început a zonei alocate tabloului.

Accesarea unui element al un tablou se implementează de regulă utilizând tehnica adresării indexate.

- **tipul mulţime**

Numeroşi algoritmi prelucrează colecţii de obiecte în care ordinea elementelor nu contează. În astfel de situaţii se recomandă folosirea tipului mulţime.

Tipul mulţime se defineşte pornind de la un tip de bază T şi el reprezintă toate submulţimile lui T . Un astfel de tip se mai numeşte şi mulţimea putere a lui T.

Pentru reprezentarea internă a obiectelor de tip mulţime se foloseşte de obicei funcţia caracteristică.

structura $2^{\mathcal{T}}$

operaţii

$\emptyset :\rightarrow 2^{\mathcal{T}}$

$inser : 2^{\mathcal{T}} \times \mathcal{T} \rightarrow 2^{\mathcal{T}}$

$elim : 2^{\mathcal{T}} \times \mathcal{T} \rightarrow 2^{\mathcal{T}}$

$\cdot \in \cdot : \mathcal{T} \times 2^{\mathcal{T}} \rightarrow Logic$

$\cdot \cap \cdot : 2^{\mathcal{T}} \times 2^{\mathcal{T}} \rightarrow 2^{\mathcal{T}}$

$\cdot \cup \cdot : 2^{\mathcal{T}} \times 2^{\mathcal{T}} \rightarrow 2^{\mathcal{T}}$

$\cdot \setminus \cdot : 2^{\mathcal{T}} \times 2^{\mathcal{T}} \rightarrow 2^{\mathcal{T}}$

proprietăţi

$(X \in \emptyset) = fals$

$(Y \in inser(M, X)) = (X = Y \ : \ adevarat \ ? \ Y \in M)$

$(elim(inser(M, X), Y) = (X = Y \ : \ elim(M, Y) \ ? \ inser(elim(M, Y), X))$

$elim(\emptyset, X) = \emptyset$

$inser(inser(Y, M), X) = (X = Y \ : \ inser(M, X) \ ? \ inser(inser(M, X), Y))$

$(\emptyset \cap M) = \emptyset$

$(inser(M, X) \cap N) = (X \in N \ : \ inser(M \cap N, X) \ ? \ M \cap N))$

$(\emptyset \cup M) = M$

$(inser(M, X) \cup N) = inser(M \cup N, X)$

$(\emptyset \setminus M) = \emptyset$

$(inser(M, X) \setminus N) = (X \in N \ : \ M \setminus N \ ? \ inser(M \setminus N, X))$

Reprezentarea axiomatică a tipului mulţime

- **tipuri cu variante**

În practică este uneori convenabil să se considere că mai multe tipuri diferite sunt variante ale aceluiaşi tip. În acest caz se va defini un tip cu variante a cărei mulţime de valori este egală cu reuniunea disjunctă a tipurilor iniţiale. Cardinalitatea unui tip cu variante se determină ca sumă a cardinalităţilor tipurilor variante ale sale.

2.4. LISTE, STIVE, COZI

Una dintre cele mai simple şi mai des folosite structuri de date este lista liniară, cunoscută şi sub numele de listă ordonată sau secvenţă. O listă liniară constă dintr-o înşiruire de n elemente X1, X2, ..., Xn, numite noduri, aparţinând de obicei unui acelaşi tip de bază T. Proprietăţile structurale esenţiale ale acestui şir se rezumă la poziţia relativă a elementelor, aşa cum apar ele în cadrul şirului, adică:

- dacă n>0 atunci X1 este primul nod şi Xn este ultimul nod
- daca 1< i< n atunci nodul al i-lea, Xi, este precedat de nodul Xi-1 şi urmat de nodul Xi+1.

Ultima proprietate exprima esenţa accesului secvenţial, care afirmă că dacă se cunoaşte nodul curent Xi atunci se poate accesa eficient nodul predecesor Xi-1 şi nodul succesor Xi+1.

Există o mare varietate de operaţii ce pot fi realizate asupra listelor liniare. Acestea includ:

- determinarea lungimii – a numarului de elemente - unei liste
- determinarea valorii primului, respectiv ultimului element al unei liste
- descompunerea unei liste în primul element şi lista formată din restul elementelor, respectiv în ultimul element şi lista restului elementelor
- adăugarea unui element la începutul, respectiv sfârşitul unei liste
- testarea dacă lista este sau nu vidă
- parcurgerea listei de la început la sfârşit sau invers
- determinarea valorii celui de al i-lea element al listei
- modificarea valorii celui de al i-ea element al listei
- inserarea unui element pe poziţia i
- ştergerea elementului din poziţia i

Alegerea setului corespunzator de operaţii depinde de aplicaţia în care sunt folosite listele. Restrângerea setului de operaţii ne conduce la obţinerea unor noi structuri de date gen listă, dintre care unele apar foarte frecvent în aplicaţii şi din acest motiv au căpătat nume speciale:

- stiva este o listă liniară pentru care toate inserările şi ştergerile şi în general orice tip de accese au loc numai la unul dintre capetele sale. Acesta se numeşte vârful stivei, iar celălalt capat inaccesibil în mod direct se numeşte baza stivei
- coada este o listă liniară în care toate inserarile se fac la unul dintre capete, numit spate, iar ştergerile şi citirile au loc la celălalt capăt numit faţa cozii.

Listele pot fi definite recursiv astfel: o lista este fie lista vidă, fie este formată din elementul din capul listei - primul element al listei - şi lista formată din restul elementelor. Aceasta definiţie ne conduce la definiţia axiomatică din figura următoare. Am notat cu T tipul elementelor listei.

structura $Lista(\mathcal{T})$
operaţii

$$vid :\rightarrow Lista(\mathcal{T})$$
$$cons : \mathcal{T} \times Lista(\mathcal{T}) \rightarrow Lista(\mathcal{T})$$
$$cap : \{L | L \in Lista(\mathcal{T}), \neg evida(L)\} \rightarrow \mathcal{T}$$
$$rest : \{L | L \in Lista(\mathcal{T}), \neg evida(L)\} \rightarrow Lista(\mathcal{T})$$
$$evida : Lista(\mathcal{T}) \rightarrow Logic$$

proprietăţi

$$evida(vid) = adevarat$$
$$evida(cons(X, L)) = fals$$
$$cap(cons(X, L)) = X$$
$$rest(cons(X, L)) = L$$

O listă cu n elemente se poate construi prin apeluri repetate ale constructorului *cons()*. Numărul de elemente ale unei liste se numeşte lungimea listei.

Listele pot fi reprezentate folosind fie alocarea secvenţială, fie alocarea înlănţuită.

Reprezentarea listelor folosind alocarea secvenţială

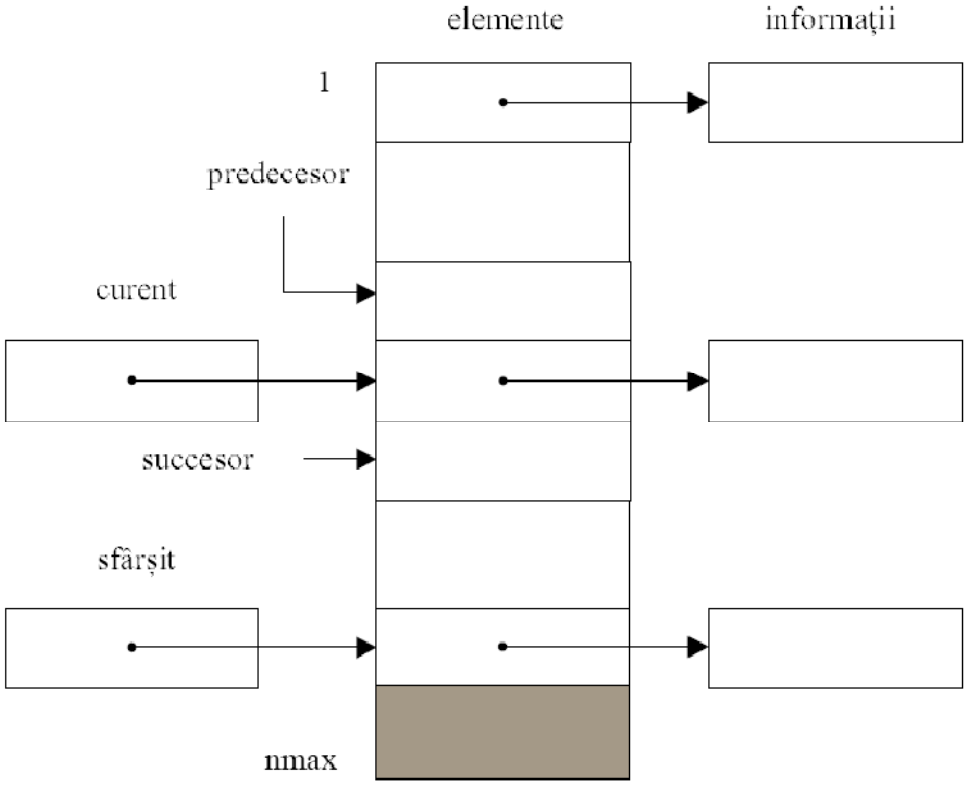
Cel mai simplu şi mai natural mod de a reprezenta în memorie o listă liniară este de a plasa elementele listei în locaţii consecutive, un nod după altul, folosind în acest scop o structură de tip tablou. O componentă a tabloului poate stoca fie un nod al listei, fie un pointer la un nod al listei.

Accesul la un element din listă, parcurgerea listei sau adăugarea/ştergerea unui element la/de la sfârşitul listei se fac cu uşurinţă, în timp ce inserarea/ştergerea de elemente din interiorul listei sunt operaţii costisitoare, doarece presupun deplasarea unor elemente.

Reprezentarea unei liste folosind alocarea secvenţială foloseşte un tablou. Alegem în cele ce urmează soluţia în care componentele acestui tablou sunt pointeri la informaţia utilă. Această metodă de reprezentare are avantajul că lista poate fi eterogenă, adică informaţiile din noduri pot aparţine unor tipuri diferite. Elementele listei vor ocupa poziţiile 1, 2, ..., din cadrul acestui tablou. În plus, trebuie cunoscută poziţia ultimului element din listă.

Pentru parcurgerea listei este necesară utilizarea unei variabile numite cursor, care ne indică poziţia elementului curent din listă (elementul care trebuie prelucrat). Dacă lista este folosită într-o aplicaţie în cadrul căreia o singură parcurgere a listei este activă la un moment dat, această variabilă poate fi adăugată la reprezentarea listei. În caz contrar, această variabilă va trebui reprezentată separat de listă. Vom presupune în continuare că ne aflăm în primul caz.

Pentru reprezentarea listei se foloseşte următoarea structură de date, care încapsulează variabila cursor, poziţia ultimului element al listei şi tabloul de pointeri în care se alocă elementele listei.

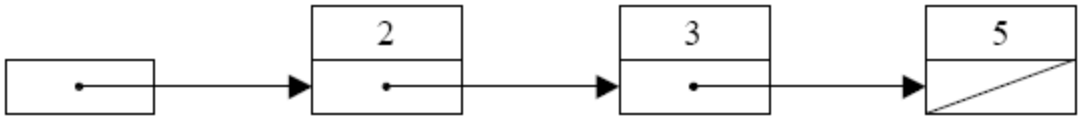


Reprezentarea listelor folosind alocarea secvenţială

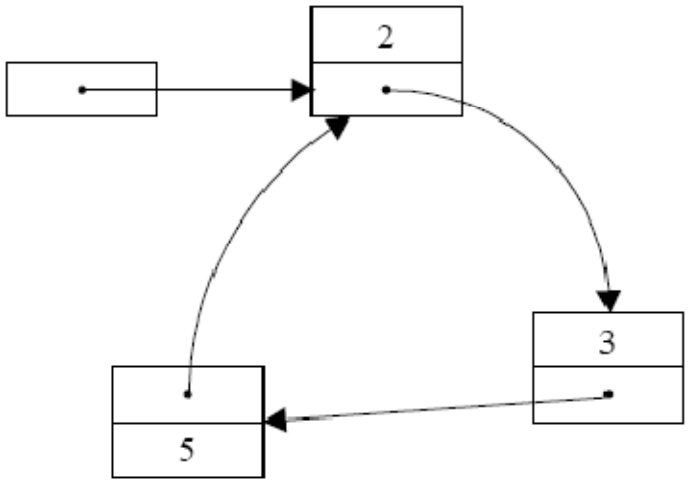
În cazul alocării înlănţuite, nodurile listei nu vor mai fi plasate în locaţii adiacente de memorie, ci în locaţii dispersate. Ordinea relativă a lor se păstrează prin includerea în cadrul fiecărui nod a unor informaţii de legătură. Spaţiul suplimentar de memorie necesar includerii informaţiilor de legătură reprezintă preţul plătit pentru înlăturarea neajunsurilor alocării secvenţiale.

O listă liniară a cărei reprezentare internă foloseşte alocarea înlănţuită a nodurilor se numeşte listă înlănţuită sau listă legată.

Un nod al unei liste poate conţine o singură legătură { către nodul succesor, caz în care vom spune că avem o listă simplu înlănţuită sau două legături { către nodurile predecesor şi succesor, caz în care vom spune că avem o listă dublu înlănţuită. Se obişnuieşte ca informaţia de legătură să fie indicată grafic printr-o săgeată orientată către informaţia "legată" de informaţia curentă. Spre exemplu, lista înlănţuită a primelor 3 numere prime este reprezentată grafic în figura următoare:



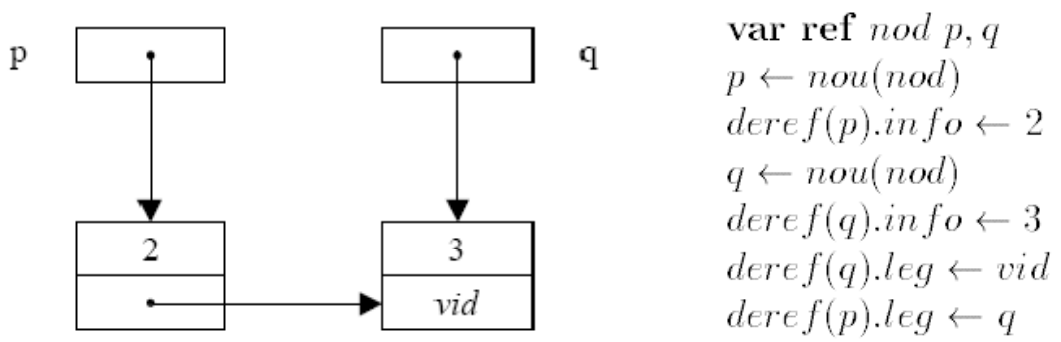
Pentru a se putea detecta sfârşitul listei, ultimul nod va conţine o legătură nulă sau echivalent vidă (reprezentată printr-o celulă "tăiată" în figură). O altă posibilitate este folosirea unei structuri în care ultimul nod să indice spre primul, ca în figura următoare. O listă înlănţuită reprezentată printr-o astfel de structură circulară se numeşte listă circulară.



Exemplu de listă circulară

Evident, putem avea liste circulare simple şi liste circulare duble.

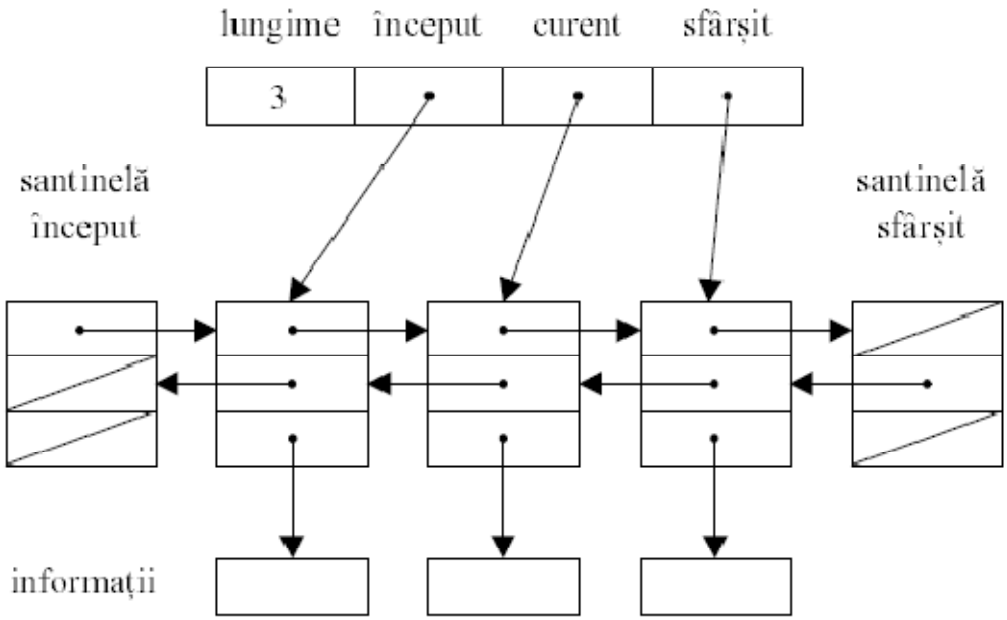
Pentru reprezentarea informaţiei de legătură se folosesc pointeri sau indici (numere întregi). În cazul în care informaţia de legătură se reprezintă prin pointeri, se obişnuieşte ca nodurile listei să fie create dinamic. Legăturile se vor completa cu valorile pointerilor, obţinute în urma operaţiilor de alocare dinamică a memoriei.



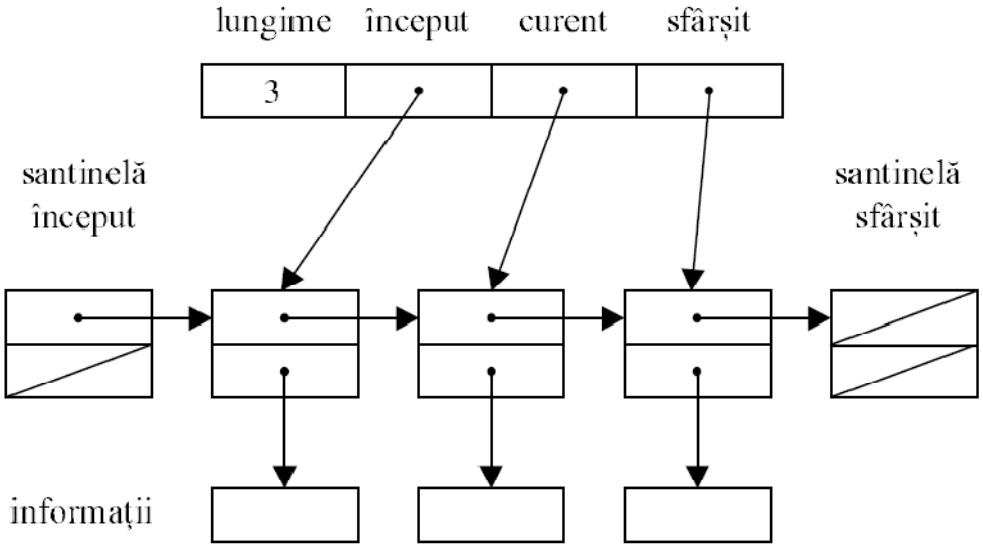
Exemplu de listă creată folosind alocare înlănţuită şi programul aferent

Dacă se realizează implementarea listelor înlănţuite folosind pointeri şi alocare dinamică, pentru reprezentarea unei liste vom folosi un obiect agregat care conţine următoarele câmpuri: lungimea listei, pointerul la primul nod, pointerul la ultimul nod şi pointerul la nodul curent (cursorul).

Pentru ca operaţiile de inserare şi ştergere să se facă la fel pentru orice nod al listei, se adaugă listei două noduri fictive numite şi santinele, plasate la începutul şi la sfârşitul listei (înaintea primului, respectiv după ultimul nod al listei). În acest fel, orice nod neredundant al listei va avea atât succesor cât şi predecesor.



Reprezentarea listelor simplu înlănţuite cu pointeri şi alocare dinamică



Reprezentarea listelor dublu înlănţuite cu pointeri şi alocare dinamică

2.5. ARBORI. REPREZENTARE, PARCURGERE

Listele sunt potrivite pentru reprezentarea datelor organizate liniar. Dacă însă dorim să descriem date structurate ierarhic, simpla enumerare a obiectelor componente cu ajutorul unor liste este insuficientă.

Organizarea datelor sub formă ierarhică este frecvent întâlnită în cele mai diverse domenii aplicative. Câteva exemple sunt: organizarea administrativă sau managerială a unei societăţi, planificarea meciurilor unei competiţii sportive de tip turneu, structurarea unei cărţi sau a directorului de fişiere dintr-un sistem de operare, reprezentarea expresiilor aritmetice şi logice într-un compilator, în vederea evaluării lor. Generalizând, orice entitate poate fi descrisă la un nivel abstract printr-un obiect primitiv sau la un nivel detaliat sub forma unei ierarhii de obiecte componente.

Pentru definirea structurilor arborescente o atenţie importantă trebuie acordată relaţiei de "ramificare" sau echivalent "subordonare".

Se numeşte arbore o mulţime A de unul sau mai multe noduri astfel încât:
i) există un nod special rad(A) € A numit rădăcina arborelui A
ii) mulţimea celorlalte noduri din A cu excepţia rădăcinii este partiţionată în $n \geq 0$ mulţimi nevide şi disjuncte Ai; $1 \leq i \leq n$, care sunt la rândul lor arbori. A1...An se numesc subarborii lui A. Dacă ordinea relativă a subarborilor A1...An în punctul ii) al definiţiei este importantă, spunem că A este un arbore ordonat. În acest caz A_i este al i-lea subarbore al lui A pentru $1 \leq i \leq n$.

Dacă pentru a distinge între doi arbori se consideră că ordinea subarborilor este nerelevantă, atunci arborii se numesc orientaţi pentru a sugera faptul că are importanţă doar orientarea arcelor (de la rădăcină spre rădăcinile subarborilor,) nu şi ordinea relativă a acestora. Arborii orientaţi sunt un caz particular de grafuri orientate. Ei trebuie deosebiţi de arborescenţe, care sunt un caz particular de grafuri neorientate.

Definirea axiomatică a arborilor cu noduri de tipul *N* este următoarea:

structura $Arb(\mathcal{N})$

operaţii

$nod2arb : \mathcal{N} \to Arb(\mathcal{N})$

$ad\ prim\ fiu : Arb(\mathcal{N}) \times Arb(\mathcal{N}) \to Arb(\mathcal{N})$

$frate : Arb(\mathcal{N}) \to Arb(\mathcal{N})$

$prim\_fiu : \{N | N \in Arb(\mathcal{N}), \neg efrunză(N)\} \to Arb(\mathcal{N})$

$grad : Arb(\mathcal{N}) \to Natural$

$rad : Arb(\mathcal{N}) \to Arb(\mathcal{N})$

$efrunza : Arb(\mathcal{N}) \to Logic$

proprietăţi

$rad(nod2arb(R)) = R$

$grad(nod2arb(R)) = 0$

$prim\_fiu(ad\_prim\_fiu(F, A)) = F$

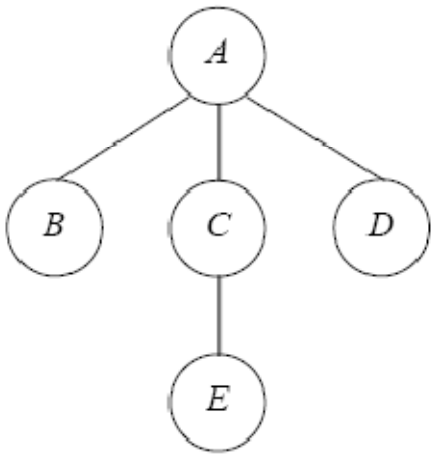
$grad(ad\_prim\_fiu(F, A)) = grad(A) + 1$

$rad(ad\_prim\_fiu(F, A)) = rad(A)$

$frate(prim\_fiu(ad\_prim\_fiu(F, A))) = (efrunza(A)\ ?\ F\ :\ prim\_fiu(A))$

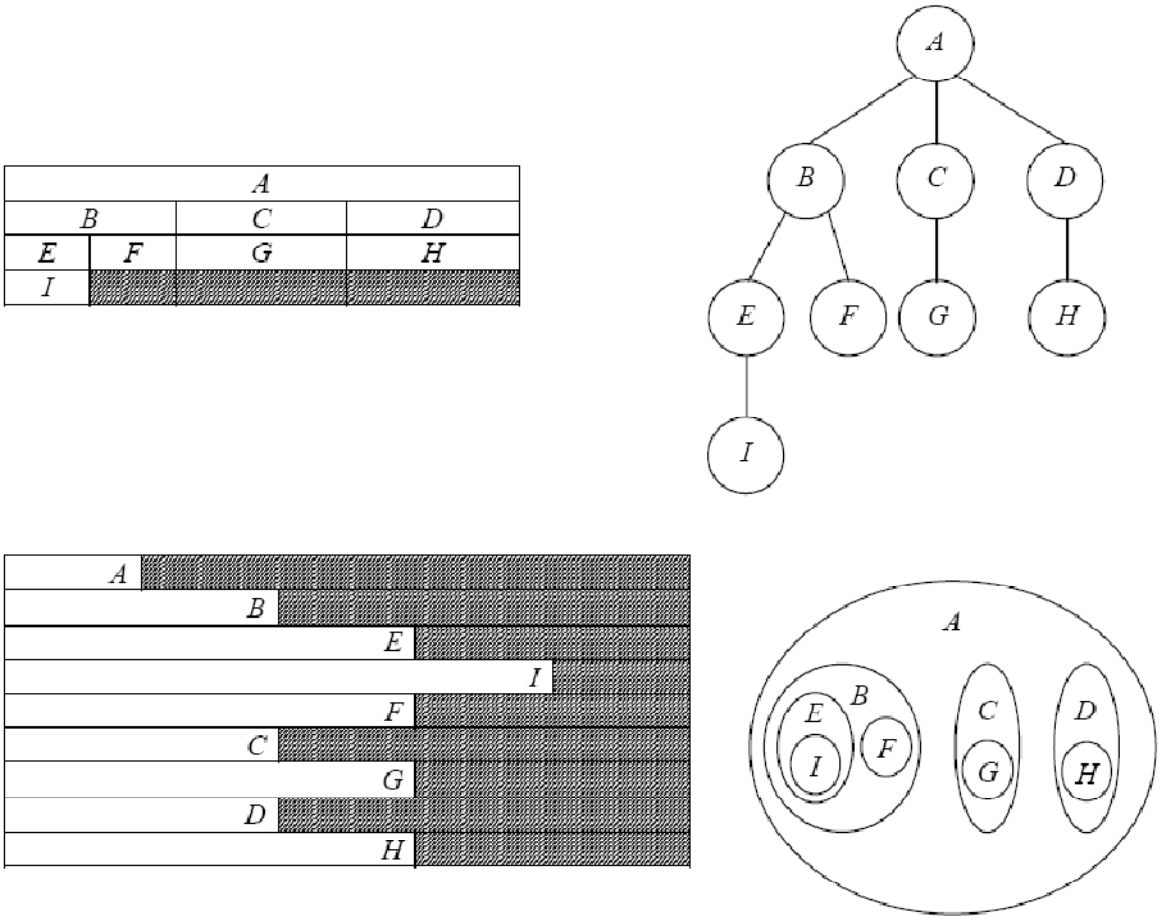
$efrunza(A) = (grad(A) = 0)$

Un exemplu de creare a unui arbore folosind această modalitate de definire este:



$$
\begin{array}{l}
ad\_prim\_fiu(\\
\quad nod2arb(B), \\
\quad ad\_prim\_fiu(\\
\quad\quad ad\_prim\_fiu(\\
\quad\quad\quad nod2arb(E), \\
\quad\quad\quad nod2arb(C) \\
\quad\quad), \\
\quad\quad ad\_prim\_fiu(\\
\quad\quad\quad nod2arb(D), \\
\quad\quad\quad nod2arb(A) \\
\quad\quad), \\
\quad), \\
)
\end{array}
$$

Structurile arborescente pot fi reprezentate grafic sau alfanumeric în diverse moduri care sugerează în fapt o aceeaşi structură.



$(A(B(E(I))(F))(C(G))(D(H))$

Figura 5.5: Reprezentări posibile ale structurilor arborescente

Numărul de subarbori ai unui arbore se numeşte gradul arborelui.

Maximul gradelor subarborilor corespunzători fiecărui nod al unui arbore se numeşte aritatea arborelui. Nodurile cu gradul egal cu 0 se numesc noduri frunză sau noduri terminale. Celelalte noduri se numesc noduri neterminale.

Rădăcinile subarborilor unui arbore cu rădăcina X se numesc copiii sau fiii nodului X. Orice nod este tatăl sau părintele fiilor săi. Copiii unui aceluiaşi nod părinte se numesc fraţi.

Pentru orice nod X al unui arbore există o cale unică ce uneşte rădăcina R a arborelui cu nodul X. Toate nodurile de pe această cale, exceptându-l pe X, formează mulţimea strămoşilor lui X. Numărul strămoşilor lui X plus 1 reprezintă nivelul nodului X. Prin înălţimea sau adâncimea unui arbore vom înţelege nivelul maxim al nodurilor sale.

O mulţime de $n \geq 0$ arbori distincţi se numeşte pădure. Dacă ordinea arborilor unei păduri este relevantă atunci avem o pădure ordonată.

Cel mai natural mod de a reprezenta o pădure este folosirea unei liste Lista(Arb(*N*)).

Se observă că prin eliminarea rădăcinii unui arbore se obţine o pădure formată din subarborii rădăcinii. Rezultă că o modalitate elegantă de a construi un arbore porneşte de la rădăcina arborelui şi pădurea subarborilor săi. Un astfel de constructor se defineşte axiomatic astfel:

cons_arb : *N* x Lista(Arb(*N*))→ Arb(*N*)

Această funcţie are proprietăţile următoare:

cons_arb(*R*, vid) = nod_2_arb(R)

cons_arb(*R*, cons(*N,L*)) = ad_prim_fiu(*N,* cons_arb(*R,L*))

Fiecare nod *N* al unei păduri se poate eticheta cu o secvenţă de numere α(N).

Mulţimea acestor secvenţe luate împreună sugerează o structură arborescentă. Dacă pădurea are k arbori, rădăcinilor li se asociază numerele 1, 2, ..., k. Dacă α este secvenţa asociată unui nod oarecare de grad m, atunci copiilor li se vor asocia secvenţele α.1, α.2, ..., α.m. Această metodă de codificare a nodurilor unei păduri se numeşte notaţia Dewey.

Spre exemplu, nodurile arborelui reprezentat axiomatic anterior se codifică astfel:

α (A) = 1

α (B) = 1.1

α (C) = 1.2

α (D) = 1.3

α (E) = 1.2.1

Un caz special de arbore frecvent întâlnit în diverse domenii aplicative este arborele binar. Într-un astfel de arbore fiecare nod are maxim doi subarbori; mai mult, când este prezent un singur subarbore, se face distincţie între subarborele stâng şi subarborele drept. Se numeşte arbore binar o mulţime finită de noduri care fie este vidă, fie constă dintr-o rădăcină şi elementele a doi arbori binari disjuncţi numiţi subarborele stâng şi subarborele drept ai rădăcinii.

Prelucrarea informaţiilor dintr-un arbore implică parcurgerea arborelui. Se numeşte parcurgere o metodă de examinare sistematică a nodurilor unui arbore astfel încât fiecare nod să fie vizitat exact o singură dată. Parcurgerea arborilor ne oferă o aranjare liniară a nodurilor, astfel încât în orice moment vom şti precis care este următorul nod care va fi vizitat şi prelucrat.

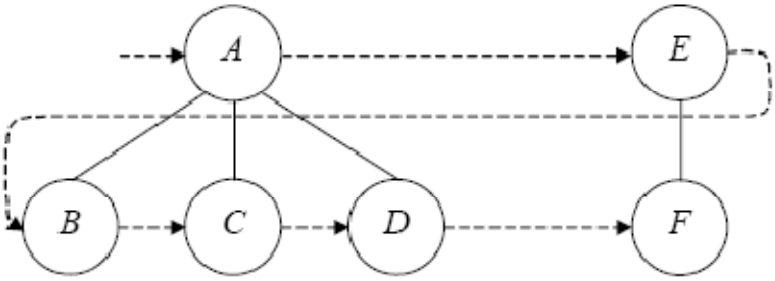
În cazul parcurgerii în lărgime, se prelucrează mai întâi informaţia din rădăcini, în ordinea în care arborii apar în cadrul pădurii, după care sunt prelucrate, de la stânga la dreapta, nodurile aflate pe primul nivel în arborii pădurii, apoi cele aflate pe al doilea nivel, ş.a.m.d. Parcurgerea în lărgime presupune folosirea unei cozi. Iniţial coada este vidă. Dacă atât coada cât şi pădurea sunt vide nu se efectuează nici o prelucrare. Dacă pădurea este nevidă, se depune primul arbore al pădurii în coadă şi parcurgerea continuă cu restul pădurii şi noua coadă. Altfel (pădurea este vidă), dacă coada este nevidă, se extrage un arbore din coadă, se vizitează rădăcina şi se continuă parcurgerea cu pădurea subarborilor arborelui extras din coadă şi cu restul cozii.

În cazul parcurgerii în adâncime, copiii unui nod sunt vizitaţi tot de la stânga la dreapta, însă trecerea de la nodul curent la fratele din dreapta are loc numai după ce a fost parcurs tot subarborele cu rădăcina în nodul curent. Pentru a memora informaţiile relative la punctul de revenire (nodul tată şi următorul fiu neprelucrat al său) se foloseşte o stivă. Operaţia de parcurgere în adâncime decurge absolut la fel ca şi cea de parcurgere în lărgime, cu deosebirea că în locul operaţiilor cu cozi se folosesc operaţii cu stive. În plus, la parcurgerea în preordine rădăcina este prelucrată înaintea parcurgerii subarborilor săi, iar la parcurgerea în postordine rădăcina este prelucrată după parcurgerea subarborilor săi.

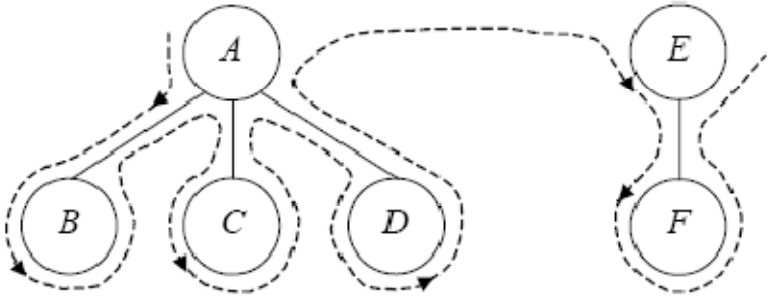
În cazul arborilor binari este consacrată parcurgerea în adâncime. Datorită faptului că un arbore binar este compus din 3 componente: rădăcina R, subarborele stâng S şi subarborele drept D, rezultă că se pot defini 3!=6 parcurgeri în adâncime, simbolizate sugestiv prin următoarele mnemonice: SRD, RSD, SDR, DSR, DRS, RDS. Dintre acestea, doar primele 3 au denumiri consacrate şi anume: inordine, preordine şi respectiv postordine. Prefixele in, pre şi post sugerează ordinea relativă de vizitare a rădăcinii în raport cu cei doi subarbori.

Arborii binari se reprezintă fie utilizând alocarea secvenţială, fie utilizând alocarea înlănţuită.

În continuare sunt prezentate grafic câteva exemple de parcurgere a arborilor.



Parcurgerea în lărgime a unei păduri



Parcurgerea în adâncime(preordine) a unei păduri

CAP.3. SISTEME DE OPERARE

3.1. NOŢIUNI INTRODUCTIVE PRIVIND SISTEMELE DE OPERARE

3.1.1 Definiţie şi rol

Se poate defini sistemul de operare, ca fiind un *set de programe specializate* ce asigură legătura funcţională între elementele componente ale unui sistem de calcul, controlând şi coordonând utilizarea resurselor acestuia între diferitele programe de aplicaţii ale utilizatorilor. El acţionează ca o interfaţă între utilizator şi hardware-ul sistemului de calcul, mascând complexitatea acestuia.

Rolul său este, pe de o parte, de a crea un mediu în care utilizatorul să poată executa programe cu mai multă uşurinţă, iar pe de altă parte, de a asigura utilizarea eficientă a hardware-ului. Aceste două obiective sunt de multe ori contradictorii.

Sistemul de operare furnizează instrumentele cu ajutorul cărora să fie folosite în mod corespunzător resursele de bază ale sistemului de calcul (hardware, software, date). Din acest punct de vedere sistemul de operare poate fi privit ca *administrator al resurselor* pe care le alocă în funcţie de necesităţi programelor şi utilizatorilor. Deoarece pot exista multe cereri (uneori conflictuale) adresate resurselor, sistemul de operare are menirea de a decide strategia de alocare, astfel încât exploatarea sistemului de calcul să fie corectă şi eficientă.

O altă definiţie a sistemului de operare, uşor diferită de cea anterioară, pune accentul pe necesitatea *controlului asupra dispozitivelor de intrare/ieşire* şi a *programelor utilizatorilor*. În această abordare, un sistem de operare este un *program de control* care urmăreşte executarea programelor utilizatorilor pentru a putea preveni eventualele erori, precum şi folosirea necorespunzătoare a sistemului de calcul şi are deci ca principală sarcină operarea şi controlul cu şi asupra dispozitivelor de I/E.

Sistemul de operare poate fi gândit ca o *maşină virtuală* interpusă între programele utilizator şi hardware furnizând astfel o maşină extinsă mult mai uşor de programat decât hardware-ul brut.

De asemenea, sistemul de operare poate fi gândit ca un *sistem care reacţionează, care răspunde*, la cereri ce sosesc de la programele utilizator (apeluri supervizor + excepţii) şi de la dispozitivele periferice (întreruperi).

3.1.2. Funcţii

Un prim *set de funcţii* pe care trebuie să le îndeplinească un sistem de operare, sunt legate de rolul său de *interfaţă între hardware şi utilizatori*. Astfel, sistemul de operare trebuie să fie capabil să asigure:

1) *posibilitatea de pregătire şi lansare în execuţie a programelor de aplicaţie*;
În acest scop, un sistem de operare trebuie să dispună de cel puţin următoarele componente:

- un *editor de texte* pentru a introduce şi modifica un *program sur*să (PS) scris într-un limbaj de programare;
- un *translator* pentru limbajul de programare folosit (asamblor, compilator sau interpretor), care să traducă instrucţiunile programului sursă, într-o formă recunoscută de sistemul de calcul (forma binară) - *program obiect* sau *module obiect* (PO);
- un *editor de legături* care să realizeze legătura dintre diverse module obiect, sau să apeleze la module obiect din bibliotecile sistemului, respectiv la modulele obiect din biblioteca utilizatorului - care au fost catalogate în prealabil, pentru a construi structură pe segmente impusă de sistemul de calcul în vederea execuţiei programelor (*program obiect executabil*- POE).

Odată construită structura pe segmente, programul va fi gata de execuţie, ceea ce va implica încărcarea acestuia în memoria internă şi execuţia efectivă; componenta sistemului de operare ce realizează acest lucru se numeşte *încărcător*.

2) *alocarea resurselor necesare execuţiei programelor* prin:
- identificarea programelor ce se execută şi a necesarului de resurse;
- alocarea memoriei interne şi a dispozitivelor periferice;
- identificarea şi protecţia colecţiilor de date.

3) *acordarea unor facilităţi prin programe utilitare de interes general*:
- gestiune cataloage (directoare, subdirectoare) şi fişiere;
- creare, modificare, copiere, mutare, ştergere, recuperare directoare şi fişiere;
- sortare/interclasare;
- depistare şi eliminare viruşi informatici şi altele.

4) *planificarea execuţiei mai multor programe* (multiprogramare) după anumite criterii, în vederea utilizării eficiente a unităţii centrale (UC);

5) *coordonarea execuţiei mai multor programe ce se execută simultan*, prin urmărirea modului de execuţie a instrucţiunilor programelor, depistarea şi tratarea erorilor, lansarea în execuţie a operaţiilor de intrare/ieşire;

6) *asistarea execuţiei programelor de către utilizator*, prin comunicaţia sistem de calcul-utilizator atât la nivel hardware cât şi la nivel software;

7) *asigurarea organizării şi proţectiei datelor în memorie*;

8) *posibilitatea generării unui sistem de operare* pe măsura configuraţiei existente.

Un al doilea *set de funcţii* pe care trebuie să le îndeplinească un sistem de operare, sunt legate de *utilizarea eficientă a resurselor sistemului de calcul*, lucru ce impune o gestionare corespunzătoare a acestora. Ca urmare, sistemul de operare trebuie să asigure:

1) *gestiunea memoriei*, care constă în:
- evidenţa acestei resurse - câtă memorie este alocată şi pentru care programe;
- cărui proces i se alocă memorie, la ce moment şi în ce cantitate - în cazul multiprogramării;
- alocarea de părţi din memorie şi asigurarea demetode de acces şi protecţie pentru procesele solicitante;
- dezalocarea zonele de memorie alocate.

2) *gestiunea procesorului*, se referă la:
- evidenţa procesoarelor şi stărilor acestora;
- decide cine poate să utilizeze procesorul, la ce moment de timp şi pentru cât timp;
- alocă procesorul la un proces prin pregătirea şi încărcarea unor registre hardware;
- retrage alocarea când procesul renunţă la utilizarea procesorului, s-a terminat sau a depăşit cuanta de timp alocată.

3) *gestiunea dispozitivelor periferice*, care constă în următoarele activităţi:
- evidenţa dispozitivelor, a unităţilor de control şi a canalelor de I/E;
- decide metoda cea mai eficientă de alocare a dispozitivelor periferice;
- dacă are loc o utilizare simultană, decide cine foloseşte resursa şi cât timp;
- alocarea dispozitivelor periferice şi iniţiază operaţia de intrare/ieşire;
- dezalocarea dispozitivelor periferice la terminarea execuţiei operaţiilor de intrare/ieşire.

4) *gestiunea informaţiei*, care se materializează în:
- evidenţierea resursei (informaţia), localizarea ei, utilizarea, starea, etc.;
- decide cine utilizează informaţia, impune protecţia cerută şi oferă rutine de acces necesare;
- alocă resursele prin deschiderea fişierului;
- dezalocă resursele prin închiderea fişierului.

3.1.3. Componentele sistemelor de operare

Din punctul de vedere al interacţiunii cu componentele hardware ale sistemului de calcul şi după modul de implementare a software-ului, sistemul de operare este organizat pe două niveluri: *fizic* şi *logic*.

Nivelul fizic oferă servicii privind lucrul cu componentele hardware ale sistemului de

calcul şi cuprinde acele elemente ce depind de configuraţia sistemului.

Nivelul logic include partea de programe prin care utilizatorului poate exploata sistemul de calcul. Comunicarea utilizatorului cu sistemul de calcul se realizează prin comenzi adresate sistemului de operare sau prin intermediul instrucţiunilor programelor pe care le execută; invers, comunicarea se realizează prin intermediul mesajelor transmise de sistemul de operare către utilizator.

Din punct de vedere funcţional, programele sistemului de operare se împart în două categorii:

- *programe de comandă şi control* (cunoscute şi sub numele de *monitoare, supervizoare* sau *executive*) - care vizează optimizarea utilizării resurselor sistemului de calcul prin coordonarea şi controlul tuturor funcţiilor acestuia;
- *programe de serviciu* - care sunt destinate optimizării accesului la resursele sistemului de calcul, fiind utilizate de programator pentru dezvoltarea programelor de aplicaţii.

Programele de comandă şi control, coordonează activitatea celorlalte componente ale sistemului de operare, îndeplinind următoarele funcţii majore:

- *gestiunea resurselor fizice şi logice* (gestiunea memoriei, gestiunea fişierelor, gestiunea fizică a intrărilor şi ieşirilor, gestiunea proceselor, evidenţa gradului de utilizare a componentelor, etc.);
- *planificarea, lansarea şi urmărirea execuţiei* (planificarea lucrărilor şi alocarea resurselor, evidenţa lucrărilor executate, a utilizatorilor şi a resurselor consumate, etc.);
- *depistarea şi tratarea evenimentelor la execuţie* (evidenţa erorilor hardware, evidenţa întreruperilor etc.).

Cele mai frecvent utilizate componente ale supervizorului sunt încărcate în memoria internă încă de la generarea sistemului de operare, fiind păstrate în aceasta pe tot parcursul execuţiei aplicaţiilor de către sistemul de calcul; aceste programe se numesc *rutine rezidente,* şi formează *nucleul* sistemului de operare; celelalte componente rămân în memoria externă fiind apelate şi executate numai atunci când sunt solicitate de către nucleu (se numesc *rutine tranziente)* asemenea oricărui program de aplicaţie (figura 3.1):

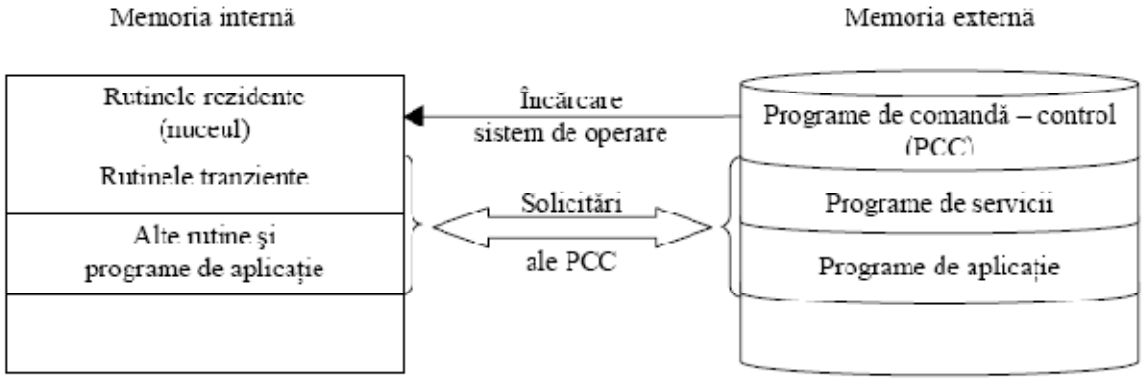


Fig. 3.1 Rutine rezidente şi rutine tranziente

Programele de serviciu se execută sub supravegherea programelor de comandă şi control şi pot fi diferite de la un sistem de operare la altul, sau chiar între versiunile aceluiaşi sistem de operare. Ele permit utilizatorului să folosească resursele fizice şi logice ale sistemului de calcul pentru realizarea aplicaţiilor. Cele mai utilizate programe din această categorie sunt:

- *Translatoarele de limbaje* - au rolul de a transforma instrucţiunile programelor sursă (scrise de utilizatori într-un limbaj de programare), în coduri executabile de calculator (format obiect). Din această categorie fac parte:
    - *asambloarele*, care au rolul de a traduce programele sursă, scrise în limbaje de asamblare, în programe obiect executabile. Aceste programe

sunt specifice unui anumit tip de sistem de calcul fiind determinate de limbajul maşină al acestuia.
    - *compilatoarele*, care sunt specifice limbajelor de programare de nivel înalt, independent de sistemul de calcul, asigurâând traducerea programelor sursă în programe obiect.
    - *interpretoarele* care analizează şi execută pas cu pas instrucţiunile programului sursă, permiţând o punere mai rapidă la punct a programelor.
- *Editorul de legături* - are rolul de a transforma modulele obiect (obţinute cu ajutorul translatoarelor) în programe executabile.
- *Bibliotecarul* - asigură crearea, gestionarea şi întreţinerea bibliotecii sistem (care conţine programele sistemului de operare) şi a bibliotecilor utilizator. Bibliotecile de programe sunt colecţii de programe organizate sub forma unor fişiere partajate în scopul utilizării lor ulterioare;
- *Programele de încărcare* - sunt componente ale programelor de serviciu, cu rolul de a încărca în memoria internă (RAM), în vederea execuţiei, programele obiect executabile;
- *Generatorul sistemului de operare* - permite utilizatorului să genereze un sistem de operare compatibil cu configuraţia hardware de care dispune (memorie internă, echipamente periferice utilizate, tipuri de interfeţe etc.) şi cu modalităţile de exploatare adoptate în funcţie de opţiunile domeniului de utilizare;
- *Depanatorul* este un program de serviciu ce oferă utilizatorului mijloace convenabile pentru depanarea şi controlul operaţiilor programului.
- *Programele de organizare a colecţiilor de date* - asigură operaţiile de intrare /ieşire prin colecţii de date (fişiere, baze de date, depozite de date);
- *Mediile de programare* - sunt programe destinate automatizării procesului de construire şi testare a programelor. Cu ajutorul mediilor de programare se realizează: editarea, compilarea şi, eventual, editarea de legături, lansarea în execuţie şi depanarea unui program;
- *Programele utilitare* - s-au dezvoltat şi diversificat odată cu perfecţionarea calculatoarelor, a modalităţilor de exploatare şi a domeniilor de aplicare, fiind adesea incluse în sistemul de operare şi destinate să extindă funcţionalitatea acestuia.

3.1.4. Moduri de operare

Pentru a proteja sistemul de calcul împotriva unei utilizări necorespunzătoare, folosirea de către programele utilizatorilor a anumitor instrucţiuni maşină este restricţionată, acestea putând fi efectuate numai de către sistemul de operare. Astfel, programele utilizator nu pot:

- să adreseze dispozitivele de I/E, direct;
- să utilizeze instrucţiunile care manipulează starea memoriei;
- să seteze bitul de mod care determină modul utilizator sau supervizor;
- să oprească calculatorul (instrucţiunea Halt).

Instrucţiunile maşină respective se numesc *instrucţiuni privilegiate*.

Pentru ca sistemul de calcul să se comporte diferit faţă de sistemul de operare pe de o parte, şi faţă de programele utilizatorilor, pe de altă parte, au fost adoptate două moduri de operare distincte, şi anume: *modul utilizator*, în cadrul căruia nu sunt utilizate instrucţiuni privilegiate şi *modul supervizor* (numit şi *mod monitor* sau *mod sistem*) în cadrul căruia toate instrucţiunile pot fi executate. Această distincţie între modurile de operare este o consecinţă a necesităţii realizării unei protecţii faţă de utilizarea necontrolată a instrucţiunilor, şi ca urmare, arhitectura oricărui sistem de operare trebuie să suporte cel puţin cele două moduri: *utilizator* şi cel *supervizor*.

Deoarece nucleul reprezintă partea protejată a sistemului de operare el rulează în modul

supervizor, protejând astfel structurile de date critice ale sistemului de operare şi regiştrii de dispozitiv împotriva programelor utilizator.

Pentru identificarea modului curent de operare se utilizează un *bit de mod* din cadrul unui registru al unităţii centrale, registru protejat. Semnificaţia bitului de mod respectiv este: 0 = supervizor, 1 = utilizator.

Transferul din modul utilizator în modul supervizor poate să se întâmple din cauza execuţiei explicite a unei *instrucţiuni apel supervizor* (sau sistem) (numită şi SVC sau TRAP) sau din cauza unei întreruperi hardware sau a unei excepţii. Transferul înapoi din modul supervizor în modul utilizator apare din cauza unei instrucţiuni return explicite (uzual numită REI, Return din întrerupere).

3.1.5. Întreruperi

Necesitatea *sistemului de întreruperi* devine evidentă în momentul în care studiem modul în care se execută programele. Uzual, procesorul execută instrucţiunile într-o ordine dată de următoarele reguli:

1. dacă instrucţiunea curentă este una de salt, va fi executată în continuare instrucţiunea de la adresa la care se face saltul;
2. în caz contrar, va fi executată în continuare instrucţiunea aflată în memorie la adresa imediat următoare după instrucţiunea curentă.

Ca o concluzie generală, întotdeauna o instrucţiune care face parte dintr-un program va fi urmată de o altă instrucţiune din acelaşi program. Până aici nu există nici o posibilitate de a părăsi programul aflat în execuţie decât dacă acesta se termină singur. Acest model corespunde în general cerinţelor, deoarece un program aflat în execuţie rulează în majoritatea timpului fără a ţine cont de existenţa sistemului de operare, totuşi, acesta trebuie să poată interveni în anumite situaţii bine definite, cum ar fi:

- o cerere explicită adresată de programul de aplicaţie, privind efectuarea unui anumit serviciu de către sistemul de operare, serviciu pe care aplicaţia nu-l poate efectua singură;
- o cerere de întrerupere venită din partea unui dispozitiv periferic, care poate să nu aibă legătură cu programul aflat în execuţie, dar care trebuie tratată imediat (altfel datele se pot pierde);
- o operaţie executată de UC care s-a terminat anormal (de exemplu o operaţie de împărţire la zero, depăşirea de capacitate, un acces anormal la o zonă de memorie, etc.), ceea ce indică încercarea unui program de a efectua o acţiune nepermisă.

Sistemul de operare va lăsa deci orice program să se execute fără interferenţe până la apariţia uneia din situaţiile descrise mai sus, dar în acest moment trebuie să preia imediat controlul. Soluţia este, aşa cum am precizat deja, de natură hardware şi este reprezentată de *sistemul de întreruperi*. Concret, acesta oferă posibilitatea întreruperii execuţiei programului curent, în anumite condiţii, pentru a trata o sarcină considerată ca fiind mult mai urgentă.

Fiecăreia din situaţiile prezentate mai sus îi corespunde unul tipurile de întrerupere:

- *întreruperi software* (externe);
- *întreruperi hardware* (externe);
- *excepţii* (întreruperi hardware interne).

Deoarece anumite întreruperi pot fi mai importante de cât altele ele dovedindu-se deci a fi prioritare, şi ca urmare este necesară stabilirea unei anumite ierarhii de priorităţi a acestora.

Unităţile centrale ale sistemelor de calcul sunt prevăzute cu instrucţiuni ce autorizează sau interzic întreruperile în anumite cazuri. Astfel, dacă programul aflat în execuţie nu trebuie să fie întrerupt, luarea în considereare şi tratarea întreruperilor va fi interzisă pentru a preveni perturbarea acestuia. Totuşi anumite întreruperi nu vor fi interzise, fie datorită necesităţii lor, fie datorită nivelului lor de prioritate. Exemplul cel mai reprezentativ este cazul întreruperii datorate

întreruperii tensiunii de alimentare, care nu va fi interzisă. Asemenea întreruperi sunt numite *nemascabile*.

Din contră, o întrerupere se spune că este *mascabilă* când există posibilitatea ca unitatea centrală să o ignore. Putem astfel să mascăm, la un moment dat, anumite întreruperi pentru conservarea derulării programului în curs faţă de orice întrerupere intempestivă (cu excepţia evident a întreruperilor nemascabile).

Tratarea unei întreruperi, indiferent care este cauza care a produs-o, se derulează în general de următoarea manieră:

- se recepţionează de către unitatea centrală o cerere de întrerupere internă sau externă;
- după acceptarea ei are loc sfârşitul tratării instrucţiunii în curs de execuţie;
- se salvează starea sistemului, adică se salvează conţinutul diverselor regiştri (numărătorul de program, etc.), în aşa fel încât să se poată relua execuţia programului întrerupt din starea în care se găsea la momentul întreruperii;
- se încarcă numărătorul de program cu valoarea adresei primei instrucţiuni a sub-programului (rutinei) de tratare asociat acestei întreruperi.
- sub-programul de tratare a întreruperii odată terminat provoacă restaurarea stării în care găsea sistemul la momentul luării în considerare a întreruperii.

Evident, rutinele care tratează situaţiile generatoare de întreruperi fac parte din sistemul de operare, care poate astfel rezolva problemele apărute.

Apeluri sistem (supervizor)

Una din sursele întreruperilor, prezentate anterior, o constituie solicitările formulate în mod explicit de programele de aplicaţii către sistemul de operare, pentru efectuarea anumitor servicii. De ce este însă necesar ca aceste servicii să fie implementate de către sistemul de operare şi nu pot fi lăsate în seama programelor ? În primul rând, unele operaţii uzuale (afişarea, căutarea pe disc etc.) se desfăşoară întotdeauna în acelaşi mod; deci, în loc de a scrie practic aceeaşi rutină în fiecare program, este mai economic de a o scrie o singură dată ca parte a sistemului de operare, astfel ca toate aplicaţiile să o poată utiliza. De altfel, apelul către un asemenea serviciu oferit de sistem nu se deosebeşte prea mult de apelul către o procedură sau funcţie din acelaşi program.

Pe de altă parte, o serie de acţiuni, în special accesele la dispozitivele periferice, prezintă riscuri considerabile pentru întregul sistem de calcul în cazul în care nu sunt realizate corect. Nu este deci convenabil de a permite programelor de aplicaţii să realizeze singure acţiunile din această categorie; se preferă ca activităţile de acest tip să fie îndeplinite numai prin intermediul unor rutine incluse în sistemul de operare. Pentru a pune în practică o asemenea abordare, trebuie să se poată interzice pur şi simplu realizarea anumitor operaţii de către programele de aplicaţii. Din nou este necesar un suport hardware.

În acest moment putem studia ce se întââmplă atunci când un program cere sistemului de operare furnizarea unui anumit serviciu. O asemenea cerere poartă numele de *apel sistem* (system call) şi ea reprezintă interfaţa dintre un program aflat în execuţie şi sistemul de operare. Apelurile sistem sunt disponibile atât sub forma de instrucţiuni scrise în limbaj de asamblare, cât şi ca funcţii sau subrutine apelabile în cadrul programelor scrise în limbaj de nivel înalt. Apelurile sistem pot fi generate prin intermediul unei rutine specializate, sau direct, şi ele sunt cerute în mod explicit de către un program utilizator pentru a trece din modul utilizator în modul supervizor.

Apelurile sistem creează, distrug şi utilizează diferite obiecte logice gestionate de către sistemul de operare. Procesele şi fişierele sunt cele mai importante obiecte, dar există apeluri supervizor şi pentru administrarea memoriei, efectuarea de intrări/ieşiri pe un terminal sau o imprimantă, etc.

Apelul sistem este tratat de către hardawre ca o *întrerupere software*, controlul fiind dat unei rutine a monitorului rezident, iar bitul de mod primeşte valoarea corespunzătoare modului

supervizor.

Fiecărui apel sistem îi corespunde o procedură de bibliotecă pe care programul utilizatorului poate să o apeleze. Această procedură plasează parametri de apel supervizor într-un loc predefinit cum ar fi registrele unităţii centrale apoi execută o instrucţiune TRAP (un apel de procedură protejat într-un anumit fel) pentru a activa sistemul de operare. Procedura de bibliotecă are ca scop de a masca detaliile instrucţiunii TRAP şi de a face să apară apelurile supervizor ca şi apeluri de proceduri obişnuite. Atunci când sistemul de operare preia controlul după TRAP, el verifică validitatea parametrilor şi efectuează în acest caz tratarea cerută. Când a terminat, sistemul de operare plasează un cod de stare într-un registru indicând dacă tratarea a reuşit sau a eşuat, apoi execută instrucţiunea RET (REturn fromTrap) pentru a reda controlul procedurii din bibliotecă. Această procedură redă controlul apelantului (programului utilizator) şi retrimite codul de stare ca o valoare de retur a funcţiei. Anumite valori complementare sunt de asemenea returnate în parametrii de apel.

Numărul şi tipul apelurilor sistem variază de la un sistem de operare la altul.

◊ **Excepţii**

Multe dintre erorile de programare pot fi detectate de către hardware şi sunt controlate, în mod normal, de către monitorul rezident. Dacă într-un program apare o instrucţiune nepermisă sau o greşeală de adresare a memoriei, hardware-ul va genera o întrerupere către monitorul rezident, numită *excepţie* ce va determina comutarea din modul utilizator în modul supervizor. Aşa cum s-a arătat, termenul de *excepţie* este folosit pentru a desemna o *întrerupere hardware internă*, generată de obicei ca urmare a apariţiei unei erori de program cum ar fi: încercrare efectuării unei împărţiri cu zero, utilizarea unei adrese de memorie ilegale, scrierea într-o locaţie de tip read-only, etc. Acţionând ca o întrerupere, aceasta va determina memorarea codului program asociat ultimei instrucţiuni executate şi va da controlul unei rutine de tratare a excepţiei din monitorului rezident. Înainte de a aborda următorul program, monitorul va executa în mod automat un vidaj de memorie şi de registre, care poate fi utilizat pentru depanarea programului abandonat.

3.2. PROCESE CONCURENTE

3.2.1. Noţiunea de proces

În cadrul sistemelor de operare moderne, se consideră ca unitate de lucru, *procesul secvenţial*. Acesta, definit ca entitate activă, este reprezentat de un program aflat în execuţie împreună cu datele sale şi cu toate celelalte informaţii necesare execuţei, instrucţiunile fiind parcurse una câte una, la momente de timp diferite.

Trebuie făcută deosebirea dintre proces şi program. *Procesul are un caracter dinamic*, el precizează o secvenţă de activităţi[1] în curs de execuţie, iar *programul are un caracter static*, el numai descrie textual această secvenţă de activităţi.

Procesele concurente numite şi *procese paralele*, presupun că la un moment dat, mai multe programe pot fi urmărite între punctul de începere şi terminare a execuţiei; în acest caz procesele interacţionează în două moduri:

- *indirect*, prin concurarea la aceiaşi resursă a sistemului;
- *direct*, prin utilizarea simultană a aceloraşi resurse.

[1] *Activitatea* (task) este o unitate de lucru internă creată de sistemul de operare, atunci când o etapă din lucrare este acceptată de sistemul de calcul. *Lucrarea* reprezintă o colecţie de activităţi ce se execută de către sistemul de calcul.

3.2.2 Stările unui proces

Pe durata execuţiei lor procesele trec printr-o serie de stări discrete, o stare fiind definită ca o activitate curentă a procesului. Aceste stări sunt:

- **starea de execuţie** (*running*) - un proces în starea de *execuţie* are alocate toate resursele necesare rulării, inclusiv UC-ul. În cazul unui sistem cu un sigur procesor, vom avea un singur proces activ la un moment dat în sistem. Procesul curent activ execută secvenţa sa de instrucţiuni maşină, putând cere sistemului de operare servicii ca operaţii I/E sau sincronizare printr-un semnal. Depinde de politica de planificare dacă sistemul returnează controlul acestui proces după terminarea serviciului sau alege un altul (din procesele în starea pregătit) pentru execuţie;
- **starea gata de execuţie** (*ready*) - un proces este în starea *pregătit (gata de execuţie)* când are alocate toate resursele de care are nevoie, cu execepţia UC-ului. De obicei, în această stare ajung procesele imediat după creare. Toate procesele în starea gata de execuţie aşteaptă ca sistemul de operare să le aloce UC-ul. Un modul al sistemului de operare, numit *planificator*, alege unul din aceste procese ori de câte ori UC-ul devine liber pentru a rula un nou proces.
- **starea blocată** (*blocked*) - procesul se află în aşteptarea realizării unui eveniment (de exemplu, terminarea unei operaţii de I/E).

Schimbarea stării unui proces poate fi determinată fie de procesul însuşi (de exemplu prin efectuarea unui apel sistem), fie datorită apariţiei unui eveniment extern (de exemplu când unitatea de comandă generează o întrerupere de ceas).

Să considerăm un sistem cu o singură unitate de centrală. La un moment dat aceasta poate executa un singur proces, dar mai multe procese pot fi gata de execuţie iar altele pot fi blocate. Din acest motiv se stabileşte o listă a proceselor gata de execuţie şi o listă a proceselor blocate. Lista proceselor gata de execuţie este ţinută în ordinea priorităţilor astfel încât primul proces ce va intra în prelucrarea unităţii de comandă va fi primul din listă. Lista proceselor blocate este neordonată deoarece aceste procese nu se vor debloca în ordinea din listă ci în ordinea în care se vor finaliza evenimentele ce au determinat blocarea lor. Există situaţii în care mai multe procese pot fi blocate aşteptând realizarea aceluiaşi eveniment, în acest caz fiind necesară introducerea unor priorităţi.

Când un proces este creat el este trecut la sfârşitul listei proceselor gata de execuţie. Pe măsură ce procesele intră în execuţie, lista lor se decalează către început. Al doilea proces va lua locul primului, al treilea va lua locul celui de-al doilea, ş.a.m.d. Când procesul trece în capul listei şi când unitatea de comandă a devenit disponibilă pentru a-l prelua, se spune despre proces că se găseşte în starea de tranziţie de la starea gata de execuţie la starea de execuţie. După cum am mai spus, un proces este format din cod şi date, fiind caracterizat de atribute şi stare dinamică. Atributele asociate unui proces pot fi asignate de către programator sau de către sistemul de operare şi includ prioritatea şi drepturile de acces. Prezentăm mai jos diagrama de tranziţie a stărilor proceselor (figura 3.2):

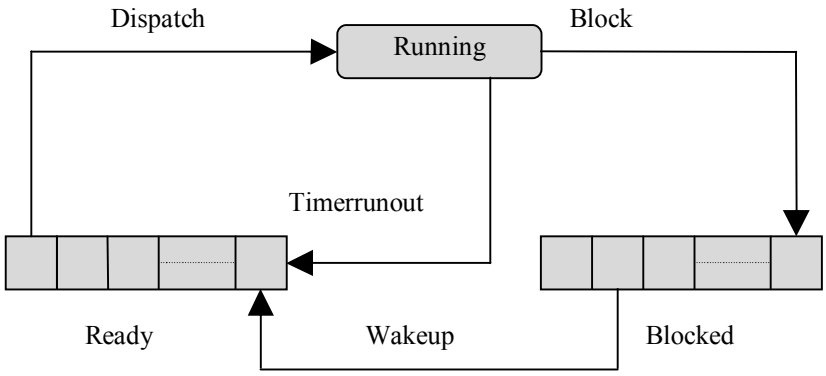


Fig 3.2. Stările de tranziţie ale unui proces

Operaţia de asignare (atribuire) a unităţii de centrale către primul proces gata din lista proceselor gata de execuţie se numeşte *expediere* (*dispatch*) şi este executată de o entitate a sistemului de operare numită *expeditor* (dispatcher). Această stare de tranziţie se indică astfel:

dispatch (nume_proces): ready ⟶ running

Pentru a preveni situaţia în care un singur proces să monopolizeze sistemul de calcul, sistemul de operare este prevăzut cu o componentă software numită *ceas de control* care permite execuţia procesului un interval de timp bine precizat. Dacă procesul nu eliberează unitatea de comandă înaintea expirării timpului, ceasul de control generează o întrerupere care permite unităţii de centrale să treacă procesul din starea de execuţie în starea gata de execuţie şi să lanseze în execuţie primul proces din lista celor gata de execuţie.

Această stare de tranziţie se indică astfel:

timerrunout (nume_proces_1): running ⟶ ready

şi

dispatch (nume_proces_2): ready ⟶ running

Dacă procesul aflat în execuţie lansează o operaţie de I/E înaintea expirării intervalului de timp ce i-a fost alocat, atunci el va trece în lista proceselor blocate, urmând ca primul proces din lista celor gata de execuţie să fie lansat în execuţie. Această stare de tranziţie o precizăm astfel:

block (nume_proces_1): running ⟶ blocked

şi

dispatch (nume_proces_2): ready ⟶ running

În cazul în care operaţia de I/E s-a terminat, procesul este trecut din starea blocat în starea gata de execuţie. Starea de tranziţie este:

wakeup (nume_proces_1): blocked ⟶ ready

Un proces ajunge în starea *suspendat* când are nevoie de o resursă pe care sistemul de operare nu i-o poate încă aloca, când invocă o rutină I/E sau aşteaptă un semnal care nu s-a produs încă. Astfel de procese ies afară din competiţia pentru UC până când condiţia de suspendare dispare. Sistemul de operare înregistrează motivul suspendării pentru a putea relua procesul mai târziu, când condiţia de suspendare dispare datorită acţiunii altor procese sau apariţiei unui eveniment extern. După apariţia evenimentului, se poate întâmpla ca UC să nu execute acest proces, întrucât în sistem există un altul cu prioritate mai mare.

La un moment dat, procesele din sistem se pot afla în diverse stări, totalitatea acestora definind *starea globală a sistemului.* Ca răspuns la acţiunile interne sau externe, procesele îşi pot schimba repede starea rezultând a nouă stare globală a sistemului.

3.2.3. Planificarea unităţii centrale

Existenţa simultană în memorie a mai multor procese face posibil ca prin intermediul unui mecanism de planificare a UC, să se îmbunătăţească eficienţa globală a sistemului de calcul, realizându-se un volum mai mare de lucru într-un timp mai scurt[2].

Procesele încărcate în memorie şi gata de a fi lansate în execuţie sunt grupate într-un şir de aşteptare (şir ready) în vederea alocării UC. Implementarea acestui şir (numit de multe ori "coadă" de aşteptare) se realizează de obicei sub forma unei liste înlănţuite ale cărei elemente sunt blocurile de control asociate proceselor (BCP), fiecare BCP incluzând un indicator (pointer) către procesul care îi urmează în şirul ready. Deoarece pe lângă UC, procesele folosesc şi alte resurse ale sistemului de calcul, pentru fiecare dintre aceste resurse pot exista şiruri de aşteptare ("cozi"), numite, de exemplu, şiruri de dispozitiv sau şiruri I/E dacă este vorba despre procesele ce aşteaptă eliberarea unui dispozitiv de I/E.

Ori de câte ori UC devine inactivă, sistemul de operare, prin intermediul unei

[2]În general, un proces foloseşte UC până în momentul în care trebuie să aştepte realizarea unei operaţii de I/E. Fără multiprogramare, pe durata realizării acestei operaţii UC rămâne inactivă. Cu multiprogramare însă, sistemul de operare permite imediat unui alt proces să folosească UC, eliminând timpul de aşteptare din primul caz.

componente numite planificator (*scheduler*), selectează pentru execuţie unul dintre procesele aflate în şirul ready. Planificatorul UC poate fi planificator pe termen lung (de perspectivă) sau planificator pe termen scurt (imediat) (figura 3.3).

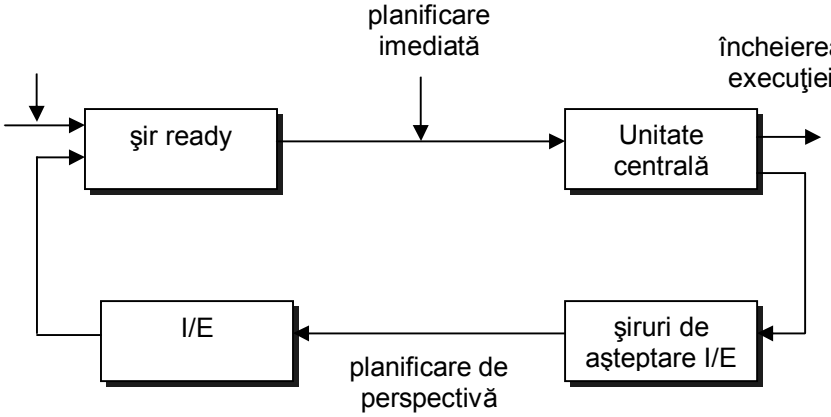


Fig. 3.3. Planificarea execuţiei proceselor

Planificatorul pe termen scurt (numit şi planificator UC) selectează unul dintre procesele gata de execuţie aflate deja în memoria internă şi îi alocă UC. Deoarece multe procese conţin cicluri "rafală" UC foarte scurte (câteva milisecunde), planificatorul pe termen scurt are o frecvenţă de execuţie foarte mare şi, prin urmare, trebuie să fie foarte rapid pentru a nu irosi timpul de lucru al UC.

Planificatorul pe termen lung (numit şi planificator de job-uri) stabileşte care sunt procesele ce vor fi încărcate în memoria internă a sistemului pentru a fi executate atunci când există mai multe cerere decât posibilităţile imediate de execuţie. De obicei, în astfel de situaţii, procesele se află memorate pe disc magnetic, de unde planificatorul pe termen lung le selectează şi le încarcă în memorie în vederea execuţiei. Cu alte cuvinte, planificatorul pe termen lung controlează gradul multiprogramării (numărul proceselor din memorie). În mod evident, frecvenţa de execuţie a acestui tip de planificator este mult mai mică şi el poate folosi mai mult timp, decât planificatorul pe termen scurt, pentru a selecta procesele. Există şi sisteme care folosesc un nivel suplimentar de planificare prin intermediul unui planificator pe termen mediu (vezi figura 3.4).

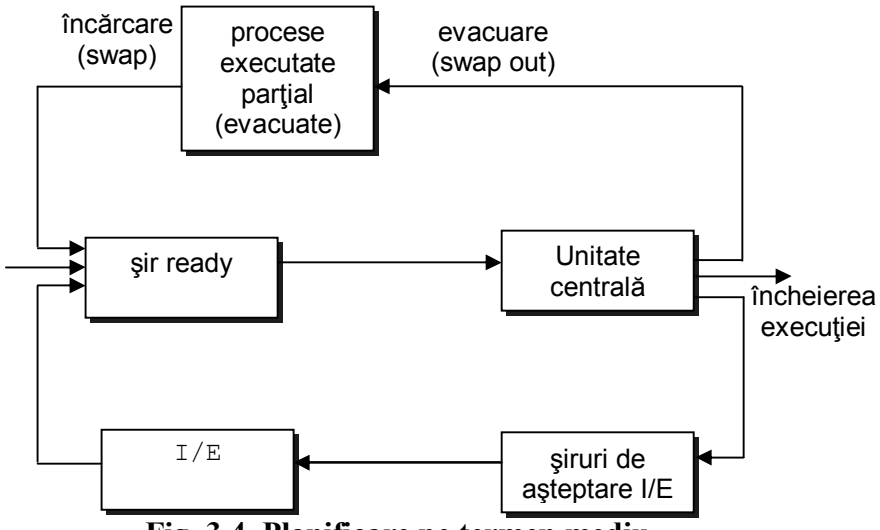


Fig. 3.4. Planificare pe termen mediu

Rolul acestuia este de a modifica gradul de multiprogramare atunci când este necesar, evacuând din memorie anumite procese (care altfel ar concura pentru dobândirea UC) şi reintroducându-le în momentul în care încărcarea sistemului permite reluarea execuţiei din

punctul în care fusese întreruptă. Pentru această metodă se foloseşte în limbajul de specialitate denumirea de *swapping* ceea ce ar însemna interschimbare sau, mai sugestiv, evacuare şi introducere din/în memorie a procesului. Planificatoarele pe termen mediu sunt incluse mai ales în cadrul sistemelor de tip time-sharing şi a celor cu memorie virtuală.

3.2.3.1 Algoritmi de planificare a UC

Prezentarea cât mai fidelă a funcţionării algoritmilor de planificare UC ar presupune utilizarea unor exemple care să includă un număr mare de procese, fiecare dintre ele fiind o secvenţă de sute de cicluri "rafală" UC şi I/E. Pentru ca toate acestea să nu afecteze înţelegerea elementelor esenţiale, s-a preferat folosirea unei variante simplificate în care fiecare proces conţine un singur ciclu "rafală" UC, iar performanţele sunt apreciate cu ajutorul duratei medii de aşteptare (notată DMA).

◊ *Algoritmul FCFS*

Algoritmul FCFS (First Come First Served) este cel mai simplu algoritm de planificare a UC: primul proces care cere alocarea UC este cel care o obţine. Şirul ready este de tip FIFO (First In First Out); atunci când un nou proces devine gata de execuţie, blocul său de control se adaugă la sfârşitul şirului ready; ori de câte ori devine disponibilă, UC este alocată procesului aflat pe prima poziţie a şirului.

Dacă în sistem există următoarele procese, sosite la acelaşi moment de timp în ordinea numerotării:

Proces	Durata ciclului "rafală"
1	10
2	29
3	3
4	7
5	12

algoritmul de planificare a UC de tip FCFS va determina executarea proceselor conform diagramei din figura următoare:

Proces 1	Proces 2	Proces 3	Proces 4	Proces 5
0 10	39	42	49	61

Duratele de aşteptare impuse fiecărui proces până la ocuparea UC sunt:

Proces	Durata de aşteptare
1	0
2	10
3	39
4	42
5	49

Durata medie de aşteptare este deci (0 + 10 + 39 + 42 + 49) : 5 = 28. Se observă că dacă procesele ar fi sosit în altă ordine (de exemplu 3, 4, 1, 5, 2), execuţia lor ar fi avut evoluţia din figura următoare:

Proces 3	Proces 4	Proces 1	Proces 5	Proces 2
0 3	10	20	32	61

durata medie de aşteptare fiind mult mai mică: (0 + 3 + 10 + 20 + 32) : 5 = 13

Exemplul ales ilustrează unul dintre dezavantajele algoritmului FCFS: durata medie de aşteptare, în general, nu este minimală şi poate varia între limite foarte largi, în funcţie de caracteristicile proceselor implicate în planificare. În plus, dacă încărcarea sistemului nu este echilibrată poate să apară aşa-numitul *efect de convoi*.

Presupunând că în sistem există un proces limitat UC şi mai multe procese limitate I/E, se poate întâmpla ca la un moment dat să se aloce UC procesului limitat UC. În timp ce acesta se execută, toate celelalte procese îşi încheie operaţiile de I/E şi intră în şirul ready, în aşteptarea eliberării UC. Pe durata aşteptării, dispozitivele de I/E stau nefolosite. În momentul în care procesul limitat UC îşi încheie ciclul "rafală" şi trece într-un şir de aşteptare I/E, toate procesele limitate I/E (care au cicluri "rafală" UC foarte scurte) se execută rapid şi intră pe rând în şirurile de aşteptare I/E. Este deci rândul UC să stea nefolosită. Apoi ciclul de funcţionare prezentat se reia, generând o utilizare ineficientă atât a UC cât şi a dispozitivelor de I/E.

◊ *Algoritmul SJF*

Algoritmul SJF (Shortest Job First - se execută mai întâi cel mai scurt job) ia în considerare pentru fiecare proces următorul ciclu "rafală" UC pe care îl conţine şi alocă UC (atunci când ea devine disponibilă) procesului cu cel mai scurt ciclu "rafală" UC următor existent în şirul ready. În cazul în care există două procese cu aceeaşi durată a ciclului "rafală" UC următor, între ele se aplică regula FCFS.

Ca exemplu, se poate folosi acelaşi set de procese utilizat în cazul algoritmului FCFS în care procesele soseau în sistem la acelaşi moment de timp, în ordinea numerotării:

Proces	Durata ciclului "rafală"
1	10
2	29
3	3
4	7
5	12

Algoritmul de planificare a UC de tip SJF, examinând durata ciclului "rafală" asociat fiecărui proces, va planifica execuţia conform figurii anterioare, rezultând o valoare a duratei medii de aşteptare de 13 unităţi de timp. Pentru comparaţie, se poate observa că algoritmul FCFS, folosind acelaşi set de procese genera o durată medie de aşteptare de 28 de unităţi de timp.

Se poate demonstra că planificând un proces scurt înaintea unuia lung, durata de aşteptare a procesului scurt se micşorează cu mai mult decât creşte durata de aşteptare a procesului lung şi, prin urmare, durata medie de aşteptare se reduce în mod substanţial. Din acest motiv, algoritml SJF este optimal, asigurând o durată medie de aşteptare minimă, oricare ar fi setul de procese luat în considerare. Singura problemă care apare este legată de cunoaşterea duratei ciclului "rafală" UC următor asociat fiecărui proces analizat.

Dacă se doreşte planificarea pe termen lung într-un sistem de tip cu prelucrare pe loturi (*batch*), se foloseşte ca valoare durata limită a job-ului precizată de către fiecare utilizator în parte[3]. Dacă însă se doreşte folosirea algoritmului pentru planificarea pe termen scurt, deoarece nu se pot cunoaşte cu exactitate duratele ciclurilor "rafală" următoare ale proceselor implicate, se poate realiza doar o aproximare a funcţionării reale.

◊ *Algoritmi bazaţi pe priorităţi*

În cadrul acestui tip de algoritmi, fiecărui proces i se asociază o prioritate (reprezentată,

[3]În cazul în care utilizatorul estimează şi precizează o valore prea mică, poate să apară eroare de depăşre a limitei de timp, fiind necesară replanificarea job-ului.

în general ca un număr cuprins într-o gamă fixată de valori), UC fiind alocată (în momentul în care devine disponibilă) procesului cu cea mai mare prioritate din şirul ready. SJF este un caz particular de algoritm bazat pe priorităţi în care prioritatea fiecărui proces este un număr invers proporţional cu mărimea ciclului "rafală" UC următor.

Prioritatea se poate defini intern sau extern. Atunci când este de tip intern, prioritatea procesului se calculează pe baza unei entităţi măsurabile cum ar fi, de exemplu, limita de timp, necesarul de memorie, numărul fişierelor deschise sau raportul dintre numărul mediu de cicluri "rafală" I/E şi numărul mediu de cicluri "rafală" UC. Dacă definirea este de tip extern, criteriile folosite sunt din afara sistemului de operare: tipul şi mărimea fondurilor rambursate pentru utilizarea calculatorului, departamentul care sponsorizează lucrarea, factorii politici, etc.

Principala problemă a algoritmilor bazaţi pe priorităţi este posibilitatea apariţiei blocării la inifinit a proceselor care sunt gata de execuţie, dar, deoarece au prioritate redusă nu reuşesc să obţină accesul la UC[4]. O astfel de situaţie poate să apară într-un sistem cu încărcare mare în care se execută un număr considerabil de procese cu prioritate ridicată; acestea vor obţine mereu accesul la UC în detrimentul proceselor cu prioritate redusă care este posibil să nu se mai execute niciodată.

O metodă de rezolvare a acestei probleme este "îmbătrânirea" proceselor, o tehnică prin care se măreşte treptat prioritatea proceselor care se constată că rămân în sistem un timp mai îndelungat. De exemplu, dacă priorităţile sunt cuprinse în domeniul 0 până la 64, se poate stabili ca la fiecare 10 minute să fie incrementat cu câte o unitate prioritatea proceselor rămase în aşteptare. În acest fel chiar şi un proces cu prioritatea iniţială 0 va reuşi ca într-un timp destul de scurt să ajungă cel mai prioritar şi, prin urmare să obţină accesul la UC (să se execute).

Algoritmi preemtivi

Toţi algoritmii de planificare a UC descrişi anterior (FCFS, SJP şi algoritmii bazaţi pe priorităţi) sunt algoritmi ne-preemtivi: odată alocată UC ea este folosită de către proces până în momentul în care acesta doreşte să o elibereze (îşi încheie execuţia sau urmează să efectueze o operaţie de I/E). Un algoritm preemtiv permite însă întreruperea execuţiei unui proces în momentul în care în şirul ready apare un alt proces cu drept prioritar de execuţie, sistemul de operare alocându-i imediat acestuia UC.

Algoritmul FCFS este prin definiţie ne-preemtiv; ceilalţi doi algoritmi prezentaţi pot fi modificaţi, astfel încât să devină preemtivi.

SJF preemtiv se formulează astfel: dacă în şirul ready soseşte un proces al cărui ciclu "rafală" UC următor este mai scurt decât ceea ce a mai rămas de executat din ciclul "rafală" UC al procesului curent, se întrerupe execuţia celui din urmă şi se alocă UC noului proces.

Un algorim preemtiv bazat pe priorităţi funcţionează într-un mod similar: ori de câte ori soseşte în şirul ready un nou proces, prioritatea sa este comparată cu cea a procesului curent; dacă se constată că este mai mare, se întrerupe execuţia procesului curent şi se alocă UC procesului nou[5].

Ca exemplu pot fi folosite următoarele trei procese, sosite în şirul ready la momentele de timp specificate:

Proces	Momentul sosirii în şirul ready	Durata ciclului "rafală"
1	0	12
2	3	3

[4]Fenomenul se mai numeşte şi "înfometarea proceselor".

[5]Dacă algoritmul bazat pe priorităţi este de tip ne-preemtiv, execuţia nu se întrerupe, noul proces fiind pus la începutul şirului ready, conform priorităţii pe care o are şi aşteptând eliberarea UC.

3	4	6

Utilizând un algoritm de planificare de tip SJF fără preemţie se obţine o evoluţie a execuţiei proceselor ca în figura următoare:

Proces 1		Proces 2	Proces 3
0	12	15	21

durata medie de aşteptare fiind: (0 + (12-3) + (15-4)) : 3 = 6,67

Dacă se foloseşte un algoritm SJF preemtiv, durata medie de aşteptare devine: (0 + (12-3) + (3-3) + (6-4)) : 3 = 3,67 evoluţia execuţiei fiind ilustrată în figura următoare:

Proces 1		Proces 2		Proces 3	Proces 1
0	3	4	6	12	21

Deoarece iniţial în şirul ready există numai procesul 1, acesta se execută până în momentul 3, când apare în şir procesul 2, a cărui durată de ciclu "rafală" (3 unităţi de timp) este mai mică decât ceea ce a mai rămas de executat din procesul 1 (9 unităţi de timp). Prin urmare, procesul 1 va fi întrerupt, UC fiind alocată procesului 2, care se va executa fără întrerupere deoarece la momentul 4, când în şirul ready soseşte şi procesul 3, necesarul duratelor de execuţie se prezintă astfel:

Proces	Necesar
1	9
2	2
3	6

În momentul încheierii execuţiei procesului 2 va fi planificat mai întâi procesul 3 şi abia după aceea procesul 1.

◊ *Algoritmul Round-Robin*

Round-Robin este un algoritm de planificare a UC proiectat special pentru sistemele time-sharing. Principalele sale caracteristici sunt definirea şi folosirea unei cunate temporale (cu valori cuprinse în domeniul 10 ms până la 100 ms) şi tratarea şirului ready ca şir FIFO circular. Planificatorul alocă pe rând UC fiecărui proces din şir pe o durată egală cu cel mult o cunată (se foloseşte un ceas setat în mod corespunzător).

Dacă durata ciclului "rafală" UC al procesului curent este mai mică decât durata cuantei, însuşi procesul eliberează UC prin emiterea unei cereri de I/E sau prin comunicarea încheierii execuţiei. Dacă însă durata ciclului "rafală" UC depăşeşte durata cuantei, ceasul va genera o întrerupere şi procesul va fi inclus la sfârşitul şirului ready, după ce în prealabil contextul său a fost salvat în blocul de control asociat.

Se poate spune deci că algoritmul Round-Robin este un algoritm preemtiv, care asigură un timp aproape egal de aşteptare pentru toate procesele din sistem. Într-adevăr, dacă în şirul ready există N procese şi se lucrează cu o cuantă de valore C, fiecare proces va folosi în total $\frac{1}{N}$ din timpul UC, fiecare alocare durând cel mult C unităţi de timp. Între două alocări succesive către acelaşi proces durata de aşteptare va fi de cel mult (N-1) x C unităţi de timp.

Ca exemplu vom fi folosit acelaşi set de procese ca şi în cazul algoritmului FCFS:

Proces	Durata ciclului "rafală"
1	10
2	29
3	3
4	7
5	12

Presupunând că se alege ca valoare a cuantei durata de 10 unităţi de timp, evoluţia execuţiei proceselor planificate cu ajutorul algoritmului Round-Robin va fi cea din figura următoare:

Proces 1	Proces 2	Proces 3	Proces 4	Proces 5	Proces 2	Proces 5	Proces 2
0 10	20	23	30	40	50	52	61

Procesul 1 foloseşte complet prima cuantă, încheindu-şi execuţia. Procesul 2 având nevoie de 29 de unităţi de timp este întrerupt în momentul expirării celei de a doua cuante, UC fiind alocată următorului proces din şirul ready. Procesul 3 se încheie înainte de epuizarea cuantei curente, eliberând UC pentru procesul 4, care are o comportare asemănătoare. În final rămân în şirul ready numai procesele 5 şi 2 care folosesc alternativ UC pe durata a încă două cuante complete şi două cuante incomplete.

Exemplul prezentat permite compararea performaţelor algoritmilor FCFS, SJF şi Round-Robin. Durata medie de aşteptare generată de planificarea setului de procese cu ajutorul algoritmului Round-Robin este: (0 + (10+20+2) + 20 + 23 + (30+10)) : 5 = 23 unităţi de timp

Reamintim că, pentru acelaşi set de procese, algoritmul FCFS generează 28 unităţi de timp, iar algoritmul SJF doar 13 unităţi de timp. Se observă că algoritmul Round-Robin asigură o valoare intermediară, în timp ce SJF este în mod evident cel mai performant.

Performanţele algoritmului Round-Robin depin în mod esenţial de mărimea cuantei folosite.

◊ *Şiruri de procese multinivel*

Atunci când procesele existente în sistem pot fi clasificate în grupe diferite, în funcţie de anumite caracteristici (de exemplu: valori ale timpului de răspuns, prioritate definită extern, necesar de memorie, etc.), se utilizează un algoritm de planificare pentru şiruri multinivel.

Şirul ready este format din mai multe subşiruri, fiecare dintre acestea conţinând câte o categorie de procese şi având propriul algoritm de planificare. De exemplu,se pot crea două subşiruri: unul pentru procese interactive (*foreground*) şi altul pentru procese de tip batch (*background*); pentru primul se poate folosi un algoritm de tip Round-Robin, iar pentru cel de-al doilea un algoritm FCFS. Este important de reţinut faptul că şi între subşiruri trebuie să existe un algoritm de planificare (de cele mai multe ori un algoritm preemtiv bazat pe priorităţi nemodificabile). De exemplu, se poate stabili ca subşirul proceselor interactive să aibă prioritate mai mare decât cel al proceselor de tip batch, ceea ce va face ca procesele din al doilea subşir să se poată executa numai atunci când primul şir este vid; dacă între timp apare un proces în primul şir, se întrerupe execuţia procesului curent şi UC este alocată noului sosit. O altă variantă de planificare între subşiruri este stabilirea a câte unei perioade de timp în care fiecare dintre acestea să deţină UC în mod exclusiv. De exemplu, pentru subşirul proceselor interactive se poate acorda 80% din timpul total al UC, iar pentru subşirul proceselor de tip batch restul de 20%.

◊ *Şiruri de procese multinivel cu feedback*

Spre deosebire de algoritmul prezentat anterior, în care procesele, odată introduse într-un subşir, rămâneau în cadrul acestuia până la încheierea execuţiei, în cazul unui algoritm de

planificare pentru şiruri multinivel cu feedback, se permite mutarea unui proces dintr-un subşir în altul, în funcţie de anumite caracteristici dinamice. Metoda favorizează procesele interactive şi procesele limitate I/E care sunt plasate în subşiruri cu prioritate ridicată, "retrogradând" procesele care folosesc UC un timp prea îndelungat în subşiruri cu prioritate redusă. De asemenea, ea previne apariţia fenomenului de "înfometare" (blocare la infinit) printr-o formă de "îmbătrânire": permite proceselor care aşteaptă de prea mult timp în şirurile de prioritate redusă să "promoveze" în cadrul şirurilor de prioritate ridicată.

Pentru a putea defini complet un algoritm de planificare pentru şirui multinivel cu feedback este necesară precizarea mai multor parametri:

- numărul de subşiruri;
- algoritmul de planificare asociat fiecărui subşir;
- metoda de stabilire a subşirului în care intră procesul atunci când doreşte să se execute;
- criteriul şi metoda de "promovare" a unui proces într-un subşir cu prioritate ridicată;
- criteriul şi metoda de "retrogradare" a unui proces într-un subşir cu prioritate redusă.

Având un înalt grad de generalitate, algoritmul de planificare pentru şiruri multinivel cu feedback este cel mai complex, oferind avantajul unor performanţe ridicate, dar necesitând în acelaşi timp un număr mare de informaţii pentru stabilirea valorilor optime în cazul parametrilor de configurare menţionaţi anterior.

3.3. GESTIUNEA MEMORIEI

3.3.1 Gestiunea în cazul multiprogramării

Organizarea memoriei cu partiţii fixe

Cea mai simplă metodă pentru administrarea memoriei în acest caz este de a o împărţi în n partiţii fixe, posibil inegale (mecanism de alocare static). O partiţie este alocată unui proces pe toată durata execuţiei lui, indiferent dacă o ocupă complet sau nu. Când soseşte un proces, el va fi pus în coada formată pentru cea mai mică partiţie suficient de mare pentru a-l cuprinde.

Deoarece partiţiile sunt fixe (au lungimi prestabilite, nemodificabile), orice spaţiu neocupat de un proces dintr-o partiţie va fi pierdut. Se produce astfel o *fragmentare internă a memoriei* (figura 3.5.):

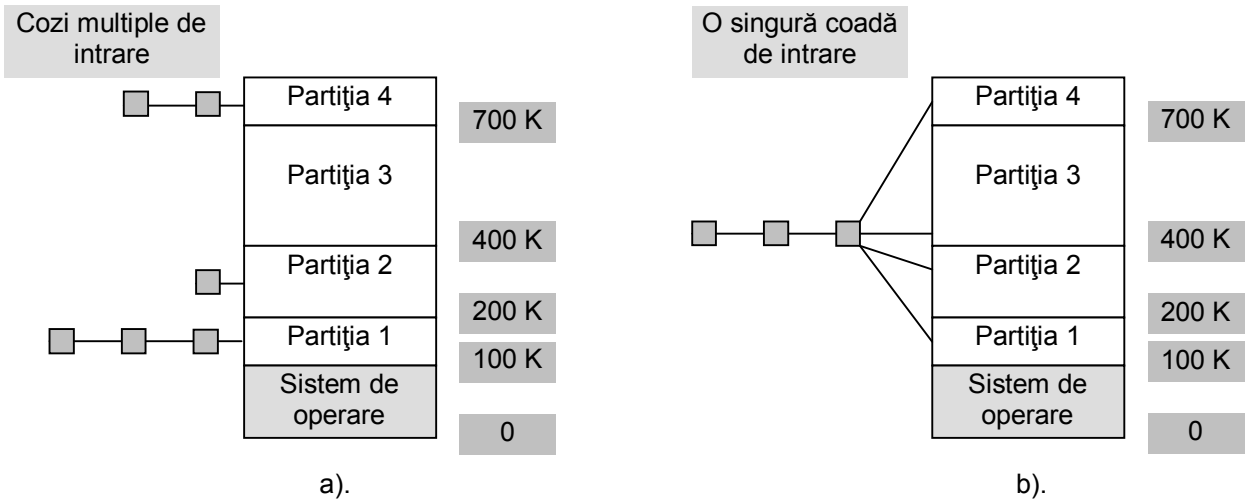


Fig. 3.5. Organizarea memoriei cu partiţii fixe

În general există două *moduri de legare* a proceselor la partiţii :

- fiecare partiţie are o coadă proprie - operatorul stabileşte de la început care sunt procesele ce vor fi executate în fiecare partiţie;

- o singură coadă pentru toate partiţiile - sistemul de operare alege, pentru procesul care urmează să intre în lucru, în ce partiţie se va executa.

Dezavantajul prezentat în figura 3.5.a) - coada pentru partiţii mari este goală (rezultă partiţie liberă, posibil de a fi alocată, dar nu există cerere), iar cea pentru partiţii mici este plină.

În cazul 3.5 b), ori de câte ori o partiţie devine liberă, primul proces din coadă care încape în acea partiţie este încărcat şi rulat. Deoarece nu se doreşte folosirea unei partiţii mari pentru un proces mic (pierdere de spaţiu), o alternativă ar fi căutarea în întreaga coadă a celui mai mare proces care se potriveşte partiţiei devenite libere. Această alternativă are dezavantajul că proceselor mici (de obicei interactive) li se acordă cel mai prost serviciu şi nu cel mai bun, cum ar fi de dorit.

O soluţie este aceea de a avea cel puţin o partiţie mică ce permite proceselor mici să ruleze fără să aştepte alocarea unei partiţii mai mari.

O altă soluţie ar fi ca un proces ce poate rula să nu fie omis de mai mult de k ori, de fiecare dată când este omis, un contor asociat lui este incrementat; iar când a ajuns la valoarea k, el nu va mai fi omis la următoarea căutare în coadă.

Multiprogramarea ridică câteva probleme esenţiale ce trebuie rezolvate (indiferent de modul de organizare a memoriei): *relocarea*, *protecţia* şi *partajarea*.

Discutăm aceste probleme relativ la organizarea memoriei în partiţii fixe.

Relocarea. Din figură se observă că procese diferite pot rula la adrese diferite, depinzâând de partiţia în care este încărcat. Dacă, de exemplu, prima instrucţiune este un apel de procedură la o adresă relativă, adresa absolută se obţine prin adunarea adresei relative cu 100 KB, dacă programul este încărcat în prima partiţie, respectiv cu 200 KB, dacă este încărcat în a doua partiţie, etc. O soluţie ar fi de a modifica adresa relativă în adresa absolută la încărcarea programului (se cunoaşte adresa de început a partiţiei). Pentru aceasta, trebuie inclus în codul programului executabil o listă sau o hartă de biţi care să specifice care cuvinte din program sunt adrese (ce trebuie relocate) şi care nu (sunt coduri de operaţii, constante, etc.).

Relocarea în timpul încărcării programului (numită şi *relocare statică*) nu rezolvă problema protecţiei. Astfel, un program poate să acceseze în scriere sau citire orice cuvâânt din memorie.

O soluţie care rezolvă relocarea este de a echipa calculatorul cu un registru special, numit *registru de bază* (conţinutul lui este păstrat în BCP-ul procesului). În acest caz avem de a face cu *relocarea dinamică*.

Astfel, când un proces este ales să ruleze, registrul de bază este încărcat cu adresa de început a partiţiei. Orice adresă de memorie generată de program este adunată cu conţinutul registrului de bază.

Un avantaj oferit de utilizarea acestui registru este acela că mutarea programului în memorie în timpul rulării presupune doar schimbarea valorii din registrul de bază. În cazul relocării în timpul încărcării, mutarea unui program presupune reluarea procesului de generare a adreselor absolute.

Protecţia. În sistemele multiutilizator, nu se acceptă ca procesele unui utilizator să scrie sau să citească datele aparţinâând altui utilizator.

Problema se pune atât la protecţia proceselor între ele cât şi la protecţia spaţiului de memorie alocat sistemului de operare, de accese neautorizate.

În sistemele ce folosesc registrul de bază pentru relocare, se utilizează un alt registru hardware numit *registru limită,* al cărui conţinut este de asemenea păstrat în BCP-ul procesului, şi care se încarcă cu lungimea partiţiei. Orice adresă generată (cu ajutorul registrului de bază) este verificată să nu depăşească partiţia curentă.

Partajarea. Pe lângă protecţie, un bun mecanism de gestionare a memoriei trebuie să asigure partajarea datelor şi codului între procesele care cooperează. Probabil că cel mai simplu mod de implementare a partajării, fără a compromite protecţia, este de a lăsa în seama sistemului de operare controlul accesului la resursele partajate. Această soluţie are unele dezavantaje: creşterea codului sistemului de operare precum şi dificultatea protejării obiectelor create

dinamic.

O soluţie este de a copia obiectul partajat în spaţiul privat (partiţia) al fiecărui proces ce îl partajează. Astfel, protecţia este asigurată. Ca urmare, actualizările se realizează doar pe copiile obiectului, nu şi pe obiectul însuşi. De asemenea, aceste actualizări trebuie propagate şi spre celelalte copii. De obicei, la fiecare comutare de context, sistemul de operare copiază datele partajate din spaţiul de adrese al procesului curent activ, în spaţiul de adrese al celorlalte procese participante.

Dezavantaje: menţinerea în memorie a mai multor copii; în cazul partajării codului, copierea lui nu are nici un sens deoarece nu are loc nici o modificare.

O altă soluţie este de a plasa datele într-un loc comun, dedicat acestui scop. În acest caz se pune problema rezolvării protecţiei: orice acces dincolo de partiţia alocată este privit ca o violare a protecţiei. Cum se rezolvă:

În sistemele cu chei de protecţie, la fiecare comutare de context se schimbă cheile tuturor blocurilor partajate cu scopul de a acorda drepturi de acces procesului curent activ. Aceasta presupune ţinerea unei evidenţe a blocurilor, care sunt partajate şi de către cine, precum şi a frecventelor modificări ale cheilor.

În sistemele ce folosesc regiştrii de bază şi limită, avem nevoie de seturi diferite de perechi de regiştri dedicaţi spaţiului de memorie privat şi respectiv celui partajat. Aceasta implică existenţa unui anumit mijloc, preferabil automat, de a desemna care este setul potrivit de regiştri pentru fiecare referire la memorie.

Cele mai multe dezavantaje ale acestei metode de organizare a memoriei se datorează partiţionării statice, sistemul devenind inflexibil, neputându-se adapta la schimbări.

O primă problemă este fragmentarea internă ce rezultă din diferenţa dintre dimensiunea partiţiei şi dimensiunea procesului ce o ocupă.

Partiţionarea fixă impune restricţii severe dimensiunii programului: nici un proces nu are voie să depăşească dimensiunea celei mai mari partiţii. Astfel, ea devine ineficientă în sistemele în care programele îşi modifică dinamic structurile de date (heap şi stivă). Un alt dezavantaj este gradul de multiprogramare limitat la numărul de partiţii.

Organizarea memoriei cu partiţii variabile

Un nou mod de organizare a memoriei care elimină multe dintre problemele create de partiţionarea fixă a memoriei este organizarea memoriei cu partiţii variabile (mecanism de alocare dinamic). În funcţie de solicitări şi de capacitatea de memorie încă disponibilă la un moment dat, numărul şi dimensiunile partiţiilor se modifică automat. Se îmbunătăţeşte factorul de utilizare a memoriei dar se complică alocarea şi dezalocarea memoriei.

În figura 3.6. sunt prezentate mai multe stări succesive ale memoriei:

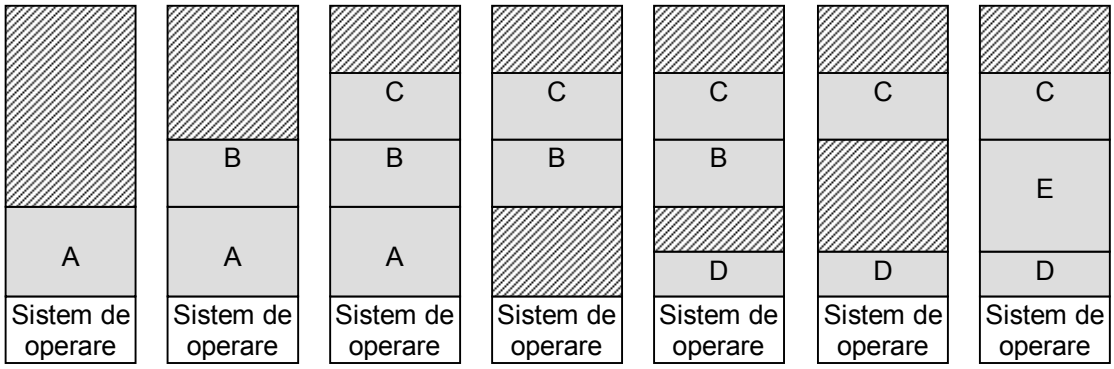


Fig. 3.6. Organizarea memoriei cu partiţii variabile

Este uşor de observat că dacă sistemul funcţionează timp îndelungat, atunci numărul spaţiilor libere va creşte, iar dimensiunile lor vor scădea, rezultând o fragmentarea externă a memoriei.

În momentul în care un proces nu are spaţiu liber de memorie în care să se încarce, sistemul de operare poate lua una din următoarele trei decizii:

1. procesul aşteaptă pâână când i se eliberează un spaţiu suficient de mare de memorie;
2. sistemul de operare încearcă alipirea unor spaţii libere vecine, în speranţa că va obţine un spaţiu de memorie suficient de mare (*cumularea golurilor*);
3. sistemul de operare decide efectuare unei operaţii de compactare a memoriei (relocare), adică de deplasare a partiţiilor active pentru a se absorbi toate zonele (fragmentele) de memorie neutilizate (*compactarea*).

Fragmentarea memoriei poate să apară din mai multe motive, unul dintre ele fiind faptul că un program şi-a terminat execuţia, iar locul rămas liber prin ieşierea acestuia din memorie nu poate fi ocupat de un alt program. Golurile în utilizarea memoriei astfel rezultate pot fi folosite de alte programe cu dimensiuni mai mici sau cel mult egale cu cele ale golului. Într-o astfel de situaţie dacă un program a încăput într-o partiţie astfel eliberată este posibil ca el să nu ocupe toată zona de memorie aferentă partiţiei, rezultând o nefolosire optimă a memoriei. Se impune deci realizarea unei operaţii de colectare a golurilor astfel apărute, adică două sau mai multe goluri adiacente să se cumuleze într-un singur gol de dimensiune mai mare. Acest proces se numeşte *cumularea golurilor*, şi el este ilustrat în figura 3.7.

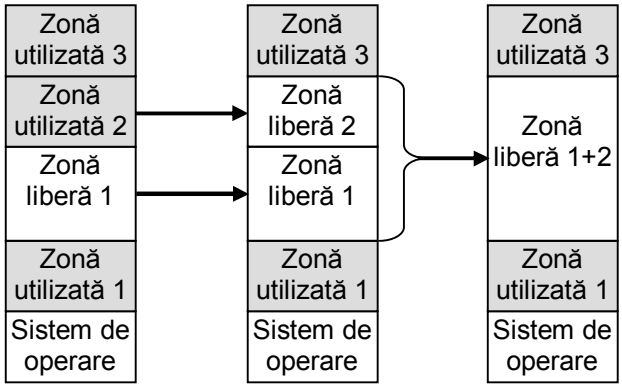


Fig. 3.7. Cumularea golurilor

Chiar în cazul în care golurile sunt colectate totuşi este posibil, ca în cadrul memoriei interne, să apară goluri dispersate. Şi în această situaţie poate apărea un job care să nu încapă în nici unul din golurile prezente. Job-ul respectiv s-ar putea rula totuşi dacă toate golurile ce sunt în mod normal dispersate ar fi cumulate într-unul singur. Realizarea acestui proces se numeşte *compactarea memoriei*, o astfel de situaţie fiind ilustrată în figura 3.8.

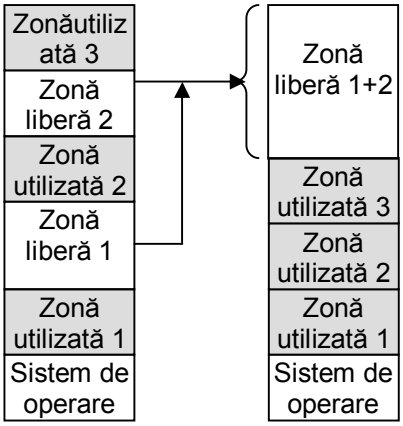


Fig. 3.8. Compactarea memoriei

De regulă, compactarea este o operaţie costisitoare şi de aceea în practică se aleg soluţii de compromis, cum ar fi :

- se lansează periodic compactarea (de exemplu, la 10 sec.), indiferent de starea sistemului. În intervalul dintre compactări, memoria apare ca un mozaic de spaţii ocupate care alternează cu spaţii libere. Procesele ce nu au loc în memorie aşteaptă compactarea sau terminarea altui proces;
- se realizează o compactare parţială pentru a asigura loc numai procesului care aşteaptă;
- se realizează compactarea ori de câte ori se eliberează o partiţie.

Deşi această schemă este cea mai bună din punct de vedere al eficienţei, totuşi ea prezintă următoarele inconveniente:

- consumarea resurselor calculatorului cu mutarea programelor dintr-o zonă în alta a memoriei;
- sistemul s-ar putea opri în timp ce se execută operaţia de compactare, ceea ce ar avea consecinţe grave pentru programele utilizator;
- compactarea consumă foarte multă memorie secundară;
- compactarea măreşte timpul de obţinere a rezultatelor rulării programelor utilizator.

Protecţia şi partajarea.

Protecţia şi partajarea, nu diferă semnificativ faţă de cazul partiţionării fixe. O diferenţă este aceea că se permite ca partiţii adiacente să se suprapună. Astfel, o singură copie a unui obiect partajat poate fi accesibil din două spaţii de adresă diferite. Partajarea, însă, este limitată la două procese. În cazul în care sunt în joc mai multe procese, trebuie aplicate metodele de la partiţionarea fixă.

Partajarea codului impune ca acesta să fie reentrant sau executat în manieră strict exclusivă, fără posibilitatea întreruperii de către alte procese. Reentranţa presupune ca variabilele să fie păstrate în stivă sau în regiştri astfel încât, în cazul întreruperii procesului curent, să nu fie afectată starea procesului întrerupt.

Dintre avantajele acestei metode amintim:

- un proces poate ocupa întreaga memorie, în cazul în care îi este necesară (nu doar partiţia cea mai mare, dacă are loc);
- în cazul extinderii unui proces dincolo de partiţia pe care o ocupă, sistemul de operare poate crea o partiţie mai mare şi poate muta procesul în ea, sau partiţia alocată procesului se poate extinde folosind zonele libere adiacente.

Partiţionarea dinamică nu este lipsită de dezavantaje. Ea consumă mult timp şi spaţiu. Fragmentarea externă poate deveni o problemă serioasă, ducând la creşterea timpului în vederea compactării. Partajarea poate ridica, de asemenea, probleme dacă obiectele partajate se supun compactării.

Se poate realiza un compromis şi anume: o parte din memorie să fie împărţită în partiţii fixe. Nucleul sistemul de operare-ului, driverele de dispozitive şi porţiuni ale sistemului de fişiere sunt buni candidaţi pentru partiţionarea fixă. Restul memoriei poate fi alocată altor procese utilizând partiţionarea variabilă.

3.3.2. Strategii de administrare a spaţiului din memoria internă

Categorii de strategii:

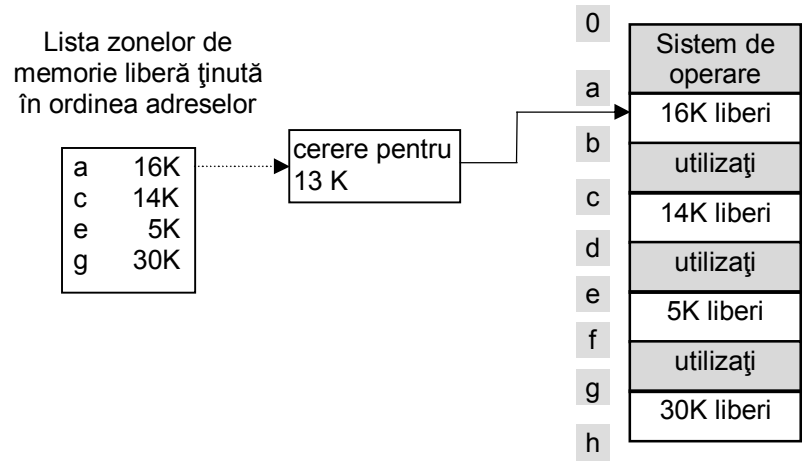
- **strategii de apel** - se bazează pe determinarea momentului când trebuie obţinută următoarea porţiune din program sau de date pentru a o transfera din memoria secundară în memoria reală. Mult timp s-a folosit *strategia de apel la cerere* în care următoarea parte de program sau de date era încărcata în memorie la momentul în care programul aflat în execuţie făcea referire la aceasta. În prezent, se tinde către utilizarea unor strategii anticipatorii, care să prevadă momentul de timp la care programul aflat în execuţie va avea nevoie de o anumită porţiune de program din

memoria secundară pentru a o încărca din timp în memoria principală, îmbunătăţind în acest fel performanţele sistemului.

- **strategii de plasare în memorie** - în general, în memorie se află "risipite" mai multe zone nealocate de diferite dimensiuni. Atunci când apare un job care necesită memorie, este aleasă una dintre acestea, cu dimensiune suficient de mare pentru a satisface cerinţele job-ului. Dacă zona este prea mare, o parte se alocă job-ului, iar restul se integrează setului de zone disponibile. În momentul terminării execuţiei, job-ul eliberează zona de memoria aferentă, care va fi inclusă şi ea în set. Dacă zona de memorie disponibilizată este adiacentă celorlate, ele se pot contopi pentru a forma o zonă de dimensiune mare, moment în care se poate verifica dacă nu cumva există job-uri aflate în aşteptare care ar putea folosi această nouă zonă. Situaţia prezentată este un caz particular al problemei de alocare dinamică a memoriei (care are ca obiect modul în care se poate satisface o cerere de dimensiune precizată având la dispoziţie *o listă de zone de memorie disponibile*), pentru care există mai multe variante de rezolvare. Deci, startegiile de plasare în memorie se concentreaza asupra determinării locului în care se va încărca un program în memoria primară.
- **strategii de înlocuire** - se ocupă cu determinarea acelei părţi din program ce va fi scoasă din memorie pentru a face loc noului program ce intră în execuţie.

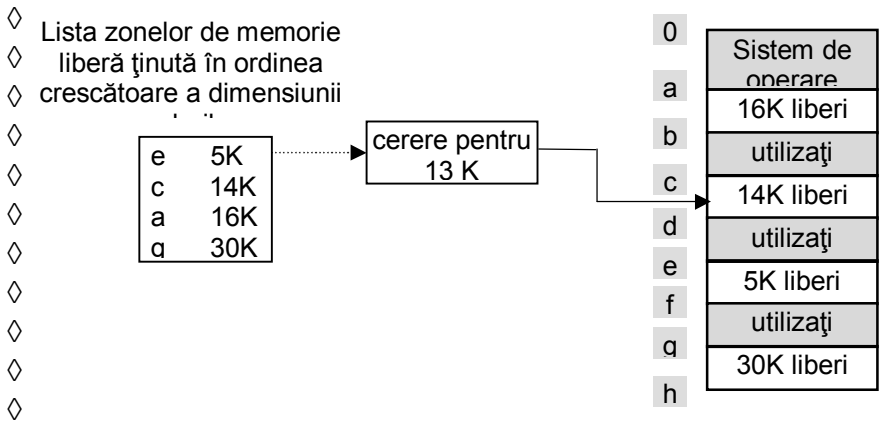
În cazul păstrării listei zonelor de memorie disponibile, ordonate după adresă, avem următorii algoritmi de alocare a memoriei (*strategii de plasare în memorie*) pentru un proces nou creat sau adus de pe disc :

First Fit (*prima potrivire*) - în cadrul acestei strategii, job-ului care vine să se execute i se va aloca prima zonă de memorie disponibilă cu dimensiune suficient de mare în care acesta poate să încapă. Căutarea zonei respective se poate face pornind fie de la începutul listei zonelor de memorie liberă, fie din oricare alt loc în care s-a terminat o căutare anterioară de acelaşi tip.

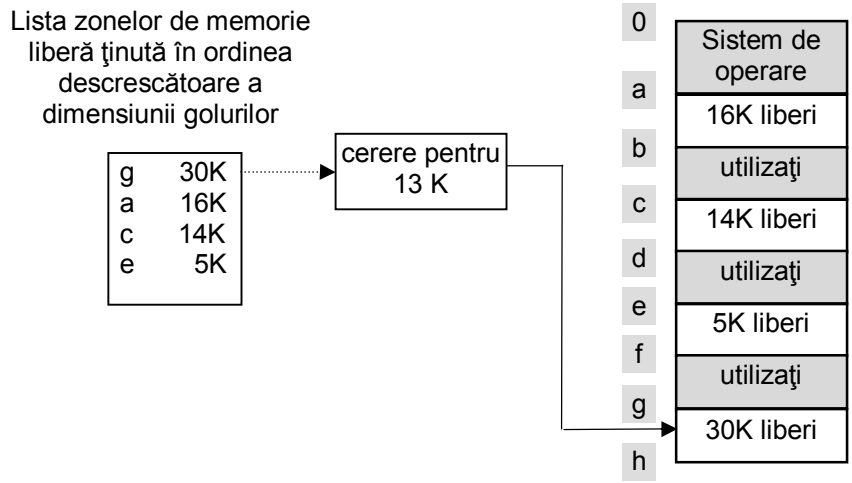


Next Fit (*următoarea potrivire*) – strategia lucrează la fel ca **First Fit**, dar căutarea porneşte de la elementul alocat în potrivirea anterioară, nu de la capătul listei.

Best Fit (*cea mai bună potrivire*) - strategia alege, dintre toate zonele de memorie disponibile a căror dimensiune permite alocarea job-ului, zona cu cea mai mică dimensiune ce poate să-l încapă, minimizându-se astfel fragmentarea internă. Lista zonelor de memorie liberă este ordonată crescător după dimensiunile golurilor.



Worst Fit (*cea mai rea potrivire*) - în cadul acestei strategii se alocă job-ului zona de memorie disponibilă care are cea mai mare dimensiune. Lista zonelor de memorie liberă este ordonată descrescător după dimensiunea golurilor.



Quick Fit (*potrivire rapidă*) - se păstrează liste diferite pentru diverse dimensiuni ce pot fi cerute. De exemplu, putem avea un tablou cu n intrări în care prima intrare este un pointer spre capul listei spaţiilor de 4K, a doua intrare spre lista de spaţii de 8K, a treia spre cele de 12K, etc. Cu acest algoritm, căutarea spaţiului de dimensiune cerută este foarte rapidă, dar se pierde mult timp pentru găsirea spaţiilor adiacente libere pentru colaţionare (altfel, memoria se umple de spaţii mici, neutilizabile).

3.3.3. Mecanismul de swapping

În mod normal, sunt mai mulţi utilizatori şi deci mai multe procese decâât pot fi păstrate în memoria principală şi de aceea este necesar păstrarea proceselor în exces, pe disc (memoria secundară). Pentru rularea acestor procese, ele trebuie aduse în memoria principală. Această mutare a proceselor pe şi de pe disc se numeşte **swapping.**

În principiu, swapping-ul se poate aplica şi în cazul organizării memoriei în partiţii fixe dar în general, acest mecanism se asociază organizării memoriei folosind partiţii variabile.

O problemă o reprezintă dimensiunea spaţiului de memorie ce trebuie alocat unui proces atunci când acesta este creat sau adus din nou în memorie prin swapping. Dacă procesele create îşi păstrează dimensiunea pe toată durata execuţiei, alocarea este foarte simplă: se alocă exact spaţiul necesar, nici mai mult, nici mai puţin.

În timpul execuţiei, însă, segmentul de date poate creşte, de exemplu, prin alocare dinamică a memoriei din heap. Dacă există un spaţiu liber adiacent procesului curent, acesta îi poate fi alocat. Dacă nu, fie trebuie mutat procesul într-un spaţiu suficient de mare, fie unul sau mai multe procese trebuie salvate pe disc pentru crearea acestui spaţiu. Dacă un proces nu are loc

în memorie pentru extindere iar spaţiul de swap de pe disc este plin, procesul este pus în starea de aşteptare sau este distrus.

Dacă este de aşteptat ca majoritatea proceselor să crească în timpul execuţiei, o idee bună este aceea de a aloca un mic extraspaţiu ori de câte ori procesul este adus în memorie de pe disc sau este mutat. În momentul salvării pe disc, doar zona de memorie utilizată va fi supusă swapping-ului.

Ori de câte ori planificatorul trebuie să lanseze în execuţie un proces şi nu este suficient loc în memorie, va apela procesul de swapping. Procesele ce se supun swapping-ului se vor alege dintre procesele suspendate care ocupă un spaţiu ce este suficient de mare pentru a fi alocat procesului curent.

În cazul operaţiei de swapping pe disc, se vor salva locaţiile ce conţin datele procesului, stiva şi regiştrii UC relevanţi, precum şi codul, în cazul sistemelor ce permit modificarea codului în timpul execuţiei.

În ceea ce priveşte spaţiul alocat pe disc swapping-ului, există două opţiuni :

- toate procesele salvate pe disc se vor depune într-un unic fişier de swap;
- fiecare proces va fi salvat separat.

În ambele situaţii, spaţiul de swap corespunzător fiecărui proces din memorie se alocă static, la crearea procesului.

O altă problemă este dacă legarea proces-partiţie se face static sau dinamic, adică dacă un proces salvat pe disc în urma unei operaţii de swap, va fi readus în memorie în acceaşi partiţie pe care o ocupa anterior swapping-ului sau în altă partiţie. Legarea statică se recomandă în cazul partiţionării statice a memoriei, eliminându-se astfel timpul afectat de către sistemul de operare alocării unei alte partiţii. Pe de altă parte, sistemele unde procesele nu sunt legate permanent de o partiţie specifică sunt mai flexibile şi duc la o utilizare mai bună a memoriei.

3.4. SISTEMUL DE OPERARE UNIX

3.4.1. Elemente introductive privind sistemul de operare UNIX

Sesiunea de lucru sub `UNIX`

După pornirea sistemului de calcul, şi încărcarea sistemului de operare, iniţierea unei sesiuni de lucru sub `UNIX` începe cu acţiunea de conectare (`login`). În cadrul acesteia, utilizatorului i se va solicita, în primă fază, identificatorul de conectare unic (`login ID`) asociat. Acesta identifică utilizatorul şi orice caracteristici asociate cu el, fiind în fapt un nume de recunoaştere autorizat. După furnizarea de către utilizator a identificatorului respectiv, i se va solicita şi parola de acces corespunzătoare (`password`). Cele două elemente sunt verificate, şi în cazul în care sunt corecte, conectarea se realizează, fiind afişate o serie de informaţii referitoare la ultima conectare nereuşită, ultima conectare reuşită, dacă sunt mesaje de poştă electronică, mesaje de la administratorul de sistem, ş.a. Apoi se lansează automat în execuţie interpretorul de comenzi implicit (`shell`)[6] afişându-se un prompter[7]. Din acest moment utilizatorul are acces la resursele sistemului, şi poate folosi comenzile `UNIX`.

[6]Interpretorul de comenzi este un program executabil situat, de regulă, în directorul /bin, şi care este tratat de către nucleul (kernel) sistemului de operare, ca orice proces utilizator neprivilegiat. La deschiderea unei sesiuni de lucru, se lansează în execuţie un proces shell la terminalul la care lucrează utilizatorul, aşteptând comenzi.
Interpretorul de comenzi (shell) (care porneşte după introducerea numelui utilizatorului şi a parolei) poate fi ales de către utilizator. Există mai multe interpretoare clasice, fiecare răspunzând anumitor cerinţe. Interpretorul "standard" în UNIX este Bourne shell (sh), dar foarte folosite sunt şi Bourne Again shell (bash), Korn shell (ksh), Berkeley C shell (csh), Turbo C shell (tcsh).

[7]În funcţie de shell-ul utilizat, prompter-ul este reprezentat printr-un caracter specific. Astfel, în cazul shell-ului sh (Bourne shell), prompter-ul este $ (în cazul utilizatorilor neprivilegiaţi) sau * (pentru utilizatori privilegiaţi - superuser, root). Dacă shell-ul este csh (C shell), prompter-ul va fi %, iar în cazul shell-ului ksh (Korn shell) prompter-ul va fi :.

Terminarea unei sesiuni de lucru UNIX se face folosind una din comenzile: exit, respectiv bye, sau tastând combinaţia de taste <Ctrl><D>.

Pentru oprirea, într-o manieră ordonată, a calculatorului pe care rulează sistemul UNIX, se utilizează comanda shutdown.

Aceasta poate fi executată numai de către utilizatorul privilegiat (superuser, root). Când comanda este executată, toţi utilizatorii primesc un mesaj shutdown şi apoi recepţionează un mesaj final de terminare.

Câţiva dintre indicatorii utilizaţi cu comanda shutdown sunt:

-h opreşte sistemul complet

-i permite afişarea de mesaje care ghidează utilizatorul în cadrul procesului de shutdown

-k simulează un shutdown sistem

-m opreşte sistemul şi îl trece în modul întreţinere (single user), astfel încât administratorul de sistem să poată efectua întreţinerea acestuia (instalarea de hardware/software sau întreţinerea planificată a hardware-ului/software-ului existent)

-r permite oprirea sistemului urmată de restartare (reboot).

Este de asemenea posibilă specificarea momentului de timp la care oprirea (sau restartarea) să fie făcută, prin indicarea unei date viitoare sau a unui timp relativ. În acest caz sistemul trimite periodic mesaje utilizatorilor, referitoare la oprire.

Variantele cele mai des folosite ale acestei comenzi sunt:

```
shutdown -h now
shutdown -r now
```

Specificarea numelui fişierelor şi directoarelor

UNIX are o structură de directoare şi fişiere asemănătoare cu cea a sistemelor DOS şi Windows.

Fişierele sunt stocate în directoare şi sunt identificate printr-un nume, care este o secvenţă de caractere.

Toate numele de fişiere, oricare ar fi tipul lor (text, binare) se supun aceloraşi reguli, şi anume:

1. Spre deosebire de sistemul DOS unde numele de fişiere era specificat în formatul 8.3, (maxim 8 caractere pentru nume şi maxim 3 caractere pentru o eventuală extensie), în UNIX se pot utiliza nume lungi de fişiere (până la 255 de caractere, dacă este instalat sistemul de fişiere ext2 sau umsdos), iar numele pot conţine mai mult de un punct (de exemplu, mihai.stud.txt);
2. În toate cazurile, în cadrul numelui unui fişier, UNIX face distincţie între caracterele minuscule şi majuscule[8], (de exemplu, numele FILENAME.tar.gz şi filename.tar.gz fac referire la fişiere diferite);
3. Caracterele ce pot fi utilizate pentru numele de fişiere sunt funcţie de sistemul UNIX folosit, existând particularităţi. În general, pot fi utilizate literele de la A la Z sau de la a la z; numerele de la 0 la 9, linia de subliniere[9], punctul[10]. În tabelul 4.1 sunt date o serie de caractere (sau combinaţii de caractere) ce au o semnificaţie specială pentru

[8]Este case-sensitive.

[9]Linia de subliniere poate separa cuvinte în cadrul numelui unui fişier, făcând numele mai uşor de citit. De exemplu, în loc de a numi un fişier fisierdetest, putem să-l numim fisier_de_test.

[10]Un punct poate fi folosit pentru a adăuga o extensie la un nume de fişier, într-o modalitatea similară fişierelor DOS. De exemplu, un fişier sursă C care conţine un program numit prog poate fi denumit prog.c. Aşa cum s-a arătat, în UNIX nu suntem limitaţi la o singură extensie. În cazul în care un punct este folosit ca prim caracter în cadrul numelui unui fişier, acesta conferă fişierului statutul de fişier ascuns (hidden). De exemplu, dacă în directorul curent există fişierele doc1 şi .doc1, tastând comanda ls va fi afişat numai fişierul doc1. Pentru afişarea ambelor fişiere se va tasta comanda ls –a.

UNIX, şi ca urmare utilizarea lor trebuie făcută cu precauţie. Lista nu este exhaustivă şi ea depinde de shell-ul UNIX folosit.

Tabelul 3.1.

Semnificaţia câtorva caractere speciale

Caracter	Semnificaţie
$	Indică începutul numelui unei variabile shell. De exemplu $var va însemna numele unei variabile numite var.
|	Leagă ieşirea standard a unei comenzi la intrarea standard a altei comenzi (pipe).
#	Începe un comentariu.
&	Execută un proces în fundal (background).
?	Specificator pentru nume de fişier global ce ţine locul unui singur caracter.
*	Specificator pentru nume de fişier global ce ţine locul mai multor caractere.
$#	Numărul de argumente transmise unui script shell.
$*	Argumente transmise unui script shell.
$?	Returnează cod de la comanda executată anterior.
>	Operator pentru redirectarea ieşirii.
<	Operator pentru redirectarea intrării.
'	Substituţie comandă.
>>	Operator pentru redirectarea ieşirii (pentru a adăuga la un fişier).
[ ]	Afişează un domeniu de caractere. [a-z] semnifică toate caracterele dintre a şi z. [a,z] semnifică caracterele a sau z.
.nume_fisier	Execută fişierul cu numele specificat.
:	Separator de nume de director din cadrul căii.

Observaţii:

1. Sub UNIX, dacă un nume de fişiere conţine spaţii (nu este recomandat, dar este posibil), ori de câte ori ne referim la el, va trebuie să includem numele respectiv între ghilimele duble.

2. Pentru fişierele executabile ("programe") nu există obligativitatea utilizării extensiilor: .COM, .EXE, .BAT. Aceste fişiere sunt marcate printr-un caracter * existent la sfârşitul numelului lor, atunci când sunt afişate cu comanda ls -F.

3. Sub DOS numele fişierelor arhivă (backup) au extensia .BAK, sub UNIX numele acestora se termină cu caracterul ~ (tilda).

Comenzi UNIX

În cadrul unui sistem UNIX utilizatorul are la dispoziţie comenzi native UNIX precum şi comenzi pe el însuşi le poate scrie (fişiere de comenzi - fişiere script). În funcţie de versiunea de UNIX ce rulează şi de shell-ul utilizat, comenzile diferă.

Comenzile UNIX sunt de fapt programe executabile care pot fi găsite în directoarele /bin, /usr/bin. Diferenţele între interpretoarele de comenzi se văd mai ales în contextul fişierelor de comenzi. Practic, aceste interpretoare permit scrierea de programe complexe, folosind comenzile UNIX şi directivele speciale.

Forma sintactică generală a unei comenzi shell este următoarea:

Comanda[indicatori][argument1][argument2]..

unde:

comanda - reprezintă numele comenzii ce urmează a fi executată. Căutarea comenzii (a fişierului executabil), până când aceasta este găsită, se face în următoarea secvenţă: în directorul curent; în directorul /bin; în directorul /usr/bin. În cazul în care comanda nu este găsită,

va fi afişat un mesaj de eroare.

[indicatori] - reprezintă opţiuni desemnate printr-o literă, precedată de semnul - sau +, ce sunt destinate pentru a obţine o rafinare a acţiunii comenzii. În cadrul unei comenzi pot fi specificaţi mai mulţi indicatori, precedaţi de unul din cele două semne menţionate anterior. De exemplu, comenzile ls -a-l şi ls -al sunt echivalente.

[argument1] [argument2] .. - reprezintă nume de fişiere/directoare sau şiruri de caractere şi constituie parametrii comenzii.

În funcţie de comandă, argumentele pot fi opţionale sau obligatorii. Specificarea argumentelor opţionale, în cadrul sintaxei unei comenzi, se face prin introducerea lor între paranteze pătrate.

Specificarea argumentelor (fişierelor) este permisă de către shell şi prin utilizarea următoarelor metacartere (specificatori pentru nume de fişier global - wildcard):

* semnifică orice şir de caractere, inclusiv şirul vid

? semnifică orice caracter

[…] semnifică o mulţime de caractere

- semnifică o secvenţă lexicografică

Pe baza acestor metacaractere, shell-ul va genera nume de fişiere utilizate în cadrul comenzilor.

Exemple:

*.c - indică toate fişierele având sufixul c;

Def[0-9] - indică fişierele def0, def1, …, def9;

Prog[09] - indică fişierele prog0, prog9;

cap? - indică fişierele existente cap1, cap2, cap3;

cap* - indică fişierele existente cap1, cap2, cap3, cap1.c, cap2.c;

prog[*?] - indică atât varianta prog*, cât şi varianta prog?.

p*r - face referire la cele care încep cu p şi se termină cu r;

c - face referire la cele care conţin litera c.

[abc]* - face referire la toate fişierele al căror nume începe cu a,b,c;

*[[I-N1-3] - face referire la fişierele ce se termină cu I,J,K,L,M,N,1,2,3;

Observaţii:

1. Cele trei construcţii lexicale comanda, [indicatori], [argument1] [argument2] .., trebuie să fie separate prin spaţii.

2. Toate comenzile acceptă intrări de la intrarea standard (de regulă, tastatura), afişează ieşirea la ieşirea standard (de regulă, ecranul terminalului).

3. UNIX oferă posibilitatea introducerii în linia de comandă a mai multor comenzi ce se vor executa serial. Comenzile trebuiesc separate prin caracterul ;.

3.4.2. Gestiunea utilizatorilor şi grupurilor

UNIX este un sistem multiuser şi multisesiune, utilizatorii putând avea deschise mai multe sesiuni de lucru pe un acelaşi calculator sau pe calculatoare diferite din reţea.

Gestiunea utilizatorilor în cauză, precum şi a grupurilor din care aceştia fac parte, reprezintă o sarcină importantă a administratorului de sistem. Pentru îndeplinirea acesteia UNIX-ul pune la dispoziţie câteva instrumente şi convenţii care fac ca această sarcină să fie mai uşor de realizat. Astfel, accesul la resursele unui sistem UNIX se realizează, de regulă, prin intermediul unor conturi utilizator ce sunt setate de către administatorul de sistem, ulterior instalării sistemului de operare. Accesul se poate realiza şi prin intermediul unor conturi sistem, dintre acestea cel mai important fiind cel aferent utilizatorului root (superuser).

Prin folosirea de conturi separate pentru fiecare utilizator, securitatea sistemului este mult mai bine asigurată, corespondenţa de unu-la-unu între utilizatori şi conturi făcând ca activitatea de gestionare a acestora să fie mai uşoară.

Contul utilizatorului `root`

Este un cont creat automat la momentul instalării sistemului de operare `UNIX`. În timp ce majoritatea conturilor utilizatorilor sunt setate astfel încât să se prevină distrugerea accidentală a sistemelor de fişiere, contul `root` nu prezintă nici o restricţie, accesul său la toate resursele sistemului, fiind deplin. Din acest motiv, folosirea acestui cont revine numai administratorului de sistem, fiind utilizat în realizarea operaţiilor de configurare şi întreţinere a sistemului `UNIX`.

Pentru prevenirea apariţiei unor incidente nedorite datorate folosirii necorespunzătoare a acestui cont, este indicată crearea unor conturi asociate unor nume de utilizator, care să aibă ca scop realizarea anumitor sarcini de administrare a sistemului, sarcini ce nu necesită un acces deplin la resursele acestuia. De exemplu, se poate seta un cont utilizator destinat efectuării copiilor de siguranţă (`backup`), un altul pentru accesul la poşta electronică, la Internet, etc.

Protecţia contului utilizatorului `root`, dar şi a celorlalte conturi utilizator, se realizează prin asignarea de parole. Alegerea acestora este indicat să se facă astfel încât depistarea lor de către ceilalţi utilizatori ai sistemului, să fie dificilă, iar schimbarea lor periodică se constituie într-o măsură de securitate obligatorie.

Nume de utilizator implicite (standard)

Aşa cum am menţionat anterior, există o serie de nume de utilizator implicite ale căror conturi sunt create în timpul procesului de instalare a sistemului de operare (în fapt este vorba de crearea fişierului `/etc/passwd`) .

Aceste nume sunt utilizate de către sistemul de operare şi administratorul de sistem, pentru realizarea unor scopuri speciale.

Fişierul `/etc/passwd`

Toate informaţiile referitore la conturile utilizatorilor sunt păstrate în fişierul `/etc/passwd`. Acest fişier se află în proprietatea utilizatorului `root` şi are identificatorul grupului (`GID`) setat la zero. Permisiunea de scriere este setată numai utilizatorului `root`, ceilalţi utilizatori având numai permisiunea de citire.

Liniile din fişierul `/etc/passwd` sunt divizate în următorul format strict:

```
nume_utilizator:parola:UID:GID:comentariu:director_implicit:coma
nda_login
```

Fiecare linie este alcătuită din câmpuri ce sunt separate prin simbolul **:**. Dacă într-un câmp nu este nimic introdus, acesta va rămâne gol, iar simbolul **:** se va păstra pentru a asigura că fiecare linie conţine şapte câmpuri. Denumirea câmpurilor (de la stânga la dreapta pentru fiecare linie) şi semnificaţia lor, este următoarea:

Câmp	Semnificaţie
`nume_utilizator`	Un identificator unic pentru utilizator
`parola`	Parola utilizatorului (criptată)
`UID`	Un număr unic care identifică utilizatorul pentru sistemul de operare
`GID`	Un număr unic care identifcă grupul la care aparţine utilizatorul
`comentariu`	Uzual, numele real al utilizatorului, sau o altă informaţie referitoare la acesta
`director_implicit`	Directorul în care utilizatorii sunt plasaţi când se conectează la sistem (director `home`)
`comanda login`	Comanda executată când utilizatorul se conectează, în mod normal un `shell`

Descrierea detaliată a câmpurilor respective este prezentată în continuare.

`nume_utilizator`

Este un câmp ce conţine un şir format din opt caractere sau mai puţin, ce identifică în mod unic fiecare utilizator. În specificarea numelui se pot folosi, pe lângă litere, şi linia de subliniere, numerele, virgula şi anumite caractere speciale. Deoarece cele mai multe comenzi `UNIX` sunt scrise cu litere mici, convenţia este ca numele de utilizator să fie şi el scris tot cu litere mici.

`parola`

În acest câmp sistemul stochează parola criptată a utilizatorului. Informaţia respectivă este foarte sensibilă la modificări necorespunzătoare, acestea putând conduce la blocarea contului utilizator căruia îi este asociată. Parola poate fi modificată numai de către administratorul de sistem (care s-a conectat cu numele de utilizator root), sau de către utilizatorul însuşi. Comanda folosită în acest scop este `passwd`.

Observaţie:

Anumite versiuni de `UNIX`, datorită unor probleme potenţiale de securitate, nu stochează parolele utilizatorilor în fişierul `/etc/passwd`.

La momentul conectării unui utilizator la sistem, parola introdusă de acesta se compară în mod logic cu un bloc de zero-uri, iar rezultatul este comparat cu intrarea corespunzătoare din fişierul de parole. Utilizatorului îi este permis accesul numai dacă cele două informaţii se potrivesc.

Câmpul de parolă este folosit pentru restricţionarea accesului la sistem. Astfel, dacă se doreşte ca un cont să nu poată fi folosit pentru accesarea sistemului, se plasează în câmpul de parolă corespunzător lui, un asterisc. În exemplul prezentat anterior, se poate observa că multe conturi de utilizator standard au un asterisc în câmpul de parolă, lucru ce efectiv blochează accesul.

În situaţia în care acest câmp nu conţine nimic (este gol), la contul respectiv se permite un acces nerestrictiv. Prin urmare, oricine poate folosi numele de utilizator respectiv pentru a i se acorda accesul imediat, fără a i se solicita introducerea unei parole.

Observaţie:

Nu se va completa câmpul de parolă cu o parolă (editându-se fişierul `/etc/passwd`) deoarece în acest fel, nu este posibilă recrearea parolei criptate, iar contul utilizatorului respectiv se va bloca.

`UID`

Fiecare nume de utilizator are asociat un identificator unic (`UID`). Acesta este folosit de către sistemul de operare `UNIX` pentru a identifica anumite informaţii ce sunt asociate cu utilizatorul. Folosirea `UID` în locul numelui utilizatorului este preferabilă, deoarece cu numerele se lucrează mai uşor decât cu caracterele, şi în plus ocupă şi mai puţin spaţiu[11].

Numerele `UID` sunt în general asignate într-un domeniu specific. Cele mai multe sisteme `UNIX`, de exemplu, alocă numerele de la `0` la `99` pentru conturile utilizator standard, şi numere `UID` de la `100` în sus pentru ceilalţi utilizatori. Asignarea numerelor `UID` pentru utilizatorii obişnuiţi ai sistemului, este bine să se facă secvenţial.

`GID`

Identificatorul grupului, `GID`, este un număr folosit pentru a păstra urma grupului la care utilizatorii aparţin când ei se conectează la sistem (grup de `startup`). Numerele `GID` se

[11]De exemplu, sistemul de operare Linux păstrează urma tuturor proceselor lansate în execuţie de către utilizator, folosind UID şi nu numele acestuia.

situează într-un domeniu ce începe cu 0 ş.am.d., numărul GID 100 fiind asignat grupului users ce conţine toţi utilizatorii neprivilegiaţi ai sistemului.

comentariu

Acest câmp este folosit de către administratorul de sistem pentru adăugarea oricărei informaţii pe care acesta o consideră necesară în identificarea utilizatorului. Tipic, această zonă este folosită pentru a introduce numele complet al utilizatorului, cu toate că anumiţi administratori de sisteme preferă să adauge alte informaţii (de exemplu numele departamentului unde lucrează utilizatorul, sau numărul de telefon al acestuia).

Anumite comenzi ale sistemului de operare pot să folosească acest câmp pentru a afişa diverse informaţii despre utilizatori, prin urmare conţinutul său poate fi alterat.

directorul_implicit (home directory)

Acest câmp specifică directorul în care va fi comutat utilizatorul după ce acesta s-a conectat la sistem (directorul curent). Fiecare utilizator din sistem trebuie să aibă dedicat un director implicit propriu[12], variabila de mediu HOME din cadrul fişierelor de startup fiind iniţializată cu această valoare. Directoarele implicite ale utilizatorilor sunt localizate într-un director comun, numit /home, ce este creat la momentul instalării sistemului Linux. Calea aferentă directoarele implicite găsite aici este de forma: /home/nume_utilizator.

Alte versiuni de UNIX folosesc ca directoare implicite directoarele /usr sau /u .

comanda_login

Reprezintă comanda ce va fi executată când procedura de conectare se termină. În cele mai multe cazuri se lansează în execuţie o comandă shell, cum este csh sau bsh, pentru a furniza utilizatorului un interpretor de comenzi (C Shell , respectiv Bourne Shell).

Dacă câmpul este lăsat gol, sistemul de operare lansează ca implicit shell-ul Bourne. Schimbarea shell-ului ce va fi lansat poate fi realizată cu ajutorul comenzilor chsh sau passwd –s. Indiferent de comanda ce este folosită, verificarea faptului că aceasta este permisă se face prin căutarea ei în fişierul /etc/shells. Numai comenzile care se află în acest fişier sunt permise ca şi intrări valide. Dacă sistemul foloseşte fişierul /etc/shells, trebuie verificat ca permisiunile de acces asupra sa şi proprietarul să fie aceleaşi ca ale fişierului /etc/passwd, altfel un utilizator îşi va putea modifica comanda_login.

Observaţii:

1. În cazul în care se doreşte invalidarea temporară a unui cont utilizator se va plasa un asterisc ca prim caracter al parolei criptate. Nu se va altera nici un caracter al parolei existente, ci numai se va adăuga un asterisc în faţa acesteia. Când se doreşte reactivarea contului se va elimina asteriscul introdus anetrior.

2. Din cauza unor potenţiale probleme de securitate, anumite versiuni de UNIX nu păstrează parolele în fişierul /etc/passwd. Dacă în câmpul de parolă din cadrul tuturor liniilor ce compun acest fişier, întâlnim caracterul x, atunci pentru stocarea parolelor este folosit un alt fişier, numit shadow password file, situat în directorul /etc/shadow, şi care poate fi citit numai de către utilizatorul root.

Grupuri

Fiecare utilizator al unui sistem UNIX aparţine unui grup. Un grup este o colecţie de utilizatori realizată în virtutea faptului că membrii ei au o caracteristică comună (de exemplu, pot să lucreze toţi în cadrul aceluiaşi departament, pot avea acces la un set de programe particular, pot avea acces la un dispozitiv special cum este un scanner sau o imprimantă laser, etc.) În

[12]În cadrul sistemului Linux, acesta poartă chiar numele utilizatorului.

general utilizatorii ce aparţin unui grup, au aceleaşi permisiuni de acces asupra unor fişiere şi directoare particulare.

Utilizatorii pot aparţine mai multor grupuri, dar la un moment dat, un utilizator poate fi membru numai al unui grup (grup primar). Aceasta deoarece la orice moment de timp, este permis un GID per utilizator.

Datorită faptului că grupurile pot avea setul lor de permisiuni de acces asupra unor fişiere şi directoare, apartenenţa sau nu a unui utilizator la un grup anume, poate determina sau restricţiona accesul la fişierele şi directoarele respective.

Grupuri implicite (standard)

În urma procesului de instalare a sistemului de operare UNIX sunt create mai multe grupuri implicite (standard) (în fapt este vorba de crearea fişierului /etc/group).

Apartenenţa la aceste grupuri este restricţionată pentru utilizatorii obişnuiţi deoarece aceasta implică obţinerea unor permisiuni de acces care sunt aceleaşi cu cele ale utilizatorului root. Ca urmare membri acestor grupuri sunt reprezentaţi de utilizatorii impliciţi ai sistemului.

Informaţia referitoare la grupuri este stocată în fişierul /etc/group, care este similar în format cu fişierul /etc/passwd.

Fiecare grup are o linie proprie în cadrul acestui fişier, iar fiecare linie este compusă din patru câmpuri ce sunt separate prin simbolul :. Două simboluri : imediat alăturate semnifică faptul că nu a fost specificată o valoare anume pentru câmpul respectiv (câmpul este gol).

Formatul aferent fiecărei linii din cadrul fişierului /etc/group este următorul:

```
nume_grup:parola_grup:GID:utilizatori
```

unde:

nume_grup

Este un câmp ce conţine un nume unic alcătuit, în general, din opt caractere sau mai puţin (se folosesc de regulă numai caractere alfanumerice) şi care specifică numele grupului respectiv.

parola

Acest câmp, de regulă, fie este liber, fie conţine un asterisc. El poate conţine totuşi şi o parolă, pe care utilizatorul trebuie să o introducă dacă doreşte să se asocieze grupului. Deşi nu toate versiunile de UNIX folosesc câmpul în cauză, el este păstrat din raţiuni de compatibilitate cu versiunile mai vechi.

GID

Ca şi în cazul fişierului /etc/passwd, acest câmp este folosit pentru a stoca un număr ce reprezintă identificatorul grupului.

utilizatori

Este un câmp ce conţine o listă a tuturor utilizatorilor ce aparţin acelui grup.

3.4.3. Permisiunile de acces asupra fişierelor şi directoarelor

Sistemul de operare UNIX este un sistem multiuser, iar permisiunile de acces asupra fişierelor şi directoarelor reprezintă una dintre facilităţile de securitate ale acestui sistem.

În fapt, este vorba de o modalitate prin care sistemul protejează fişierele şi directoarele împotriva oricărui acces neautorizat, intenţionat sau accidental.

Toate fişierele şi directoarele UNIX au permisiuni de acces şi proprietari.

La momentul creării unui fişier sau director, utilizatorul ce le-a creat devine în mod automat proprietarul acestora (owner). Aceasta înseamnă că el are privilegiul de a schimba permisiunile (specifică cine şi ce permisiuni are) sau poate schimba proprietarul fişierului

(directorului)[13].

Schimbarea permisiunilor de acces sau a proprietarului are drept scop furnizarea unui acces mai complet asupra fişierelor şi directoarelor.

Schimbarea proprietarului unui fişier sau director se face cu comanda:

```
chown proprietar_nou nume_fisier
```

Utilizatorii (dar şi fişierele şi directoarele) aparţin grupurilor. Grupurile reprezintă o modalitate convenabilă de a furniza autorizări de acces mai multor utilizatori, dar nu oricărui utilizator din sistem. La momentul creării unui utilizator, grupul implicit al acestuia este cel ce îi poartă numele şi care este creat odată cu utilizatorul.

Comanda prin care se poate schimba grupul căruia îi aparţine un fişier (director) este:

```
chgrp grup_nou nume_fisier
```

Proprietarul fişierului trebuie să fie membru al noului grup (`grup_nou`).

Observaţie:

Grupul `users` este cel ce conţine toţi utilizatorii neprivilegiaţi ai sistemului.

Permisiunile de acces asupra fişierelor (directoarelor) acceptate de `UNIX` sunt:

- *permisiunea de citire* (`r`) - permite vizualizarea (citirea) conţinutului unui fişier. În cazul unui director permite afişarea conţinutul acestuia (utilizând, de exemplu, comanda `ls`).
- *permisiunea de scriere* (`w`) - permite modificarea, ştergerea şi redenumirea unui fişier. În cazul unui director existenţa acestei permisuni permite actualizarea, ştergerea şi redenumirea directorului.
- *permisiunea de execuţ*ie (`x`) - permite executarea fişierului prin tastarea numelui acestuia. Pentru directoare permite accesul (comutarea) în directorul respectiv, şi efectuarea de operaţii asupra fişierelor din cadrul său.

Observaţii:

1. Dacă un utilizator are permisiunea de scriere asupra unui director ce conţine fişiere, permisiunile asupra fişierelor respectiv sunt rescrise de permisiunile pe acel director.

2. Orice utilizator care are permisiunea de a citi un fişier poate să copieze acel fişier. Când un fişier este copiat, copia se află în proprietatea utilizatorului care a efectuat copierea. Acesta poate să schimbe proprietarul şi permisiunile, să editeze fişierul ş.a.m.d.

3. Ştegerea permisiunii de scriere asupra unui fişier nu permite ca acesta să fie şters.

Pentru a vizualiza permisiunile existente asupra unui fişier (director) se foloseşte comanda:

```
ls -l nume_fisier
```

De exemplu, se tastează:

```
ls -l test.txt
```

Se va afişa:

```
-rw-rw-r--1 student student 150 Dec 19 08:08 test.txt
```

Interpretarea informaţiei afişate se face astfel:

Primul caracter din şirul afişat indică tipul fişierului. Astfel, dacă acest caracter este `-` atunci este vorba de un fişier normal. Dacă caracterul este `d`, atunci este vorba de un director, iar dacă caracterul este `l` este vorba de o legătură simbolică către un alt program sau fişier din sistem[14].

Următoarele nouă caractere din şirul afişat specifică, în seturi de câte trei, permisiunile de

[13]Utilizatorul root are permisiunea de a citi, scrie, şi executa orice fişier din sistem, indiferent cine este proprietarul acestora.

[14]Sunt posibile şi alte caractere.

acces pentru următoarele trei categorii diferite de entităţi:

- proprietarul (`owner`) fişierului (directorului);
- grupul (`group`) (utilizatorii ce aparţin aceluiaşi grup ca şi proprietarul fişierului /directorului)[15];
- alţii (`others`) (utilizatorii şi grupurile altele decât proprietarul şi cei din grupul căruia îi aparţine proprietarul).

Dacă unul din caracterele ce specifică permisiunile (`r`, `w`, `x`) va apărea ca fiind înlocuit prin caracterul `-`, atunci permisiunea respectivă, pentru entitatea în cauză, nu va fi acordată.

Alte informaţii afişate de către comanda `ls -l` includ: numele fişierului, data şi timpul creării sale, mărimea acestuia.

Schimbarea permisiunilor de acces asupra fişierelor

Pentru a schimba permisiunile de acces asupra fişierelor se utilizează comanda:

```
chmod indicator nume_fisier.
```

unde `indicator` se va înlocui cu identitatea entităţii pentru care se face schimbarea de permisiune:

`u` utilizatorul care este proprietar al fişierului (`owner`)
`g` grupul căruia îi aparţine utilizatorul
`o` alţii (alţii decât utilizatorul şi grupul acestuia) (`others`)
`a` oricine (`u, g` şi `o`) (`everyone`)

urmată de tipul acţiunii ce se doreşte:

`+` adaugă o permisiune
`-` elimină o permisiune
`=` marchează permisiunea ca fiind singura acceptată

şi litera (literele) aferente permisiunilor (`r, w, x`).

Observaţii:

1. În cadrul sintaxei comenzii `chmod, indicator` se poate înlocui şi cu un număr format din trei cifre, a cărui codificare este explicată în continuare.

2. Atât timp cât suntem proprietar al unui fişier (director), sau suntem conectaţi ca utilizator privilegiat (`root`), putem să modificăm permisiunile fişierului (directorului) în orice combinaţie pentru proprietar, grup şi alţii.

Există două modalităţi de modificare a permisiunilor.

O primă modalitate este cea în care se folosesc literele. Pentru a specifica entităţile pentru care se schimbă permisiunile, se vor tasta literele `u, g, o` sau `a` în sintaxa comenzii `chmod.` Litera (literele) vor fi urmate de unul dintre semnele `+, -,` sau `=,` pentru a adăuga, şterge sau marca permisiunea ca fiind singura acceptată Semnul este urmat de litera permisiunii respective (`r, w, x`).

Exemple:

Presupunem că permisiunile iniţiale ale fişierului `test.txt` sunt:

```
-rw-rw-r--1 student student 150 Dec 19 08:08 test.txt
```

Conform permisiunilor afişate proprietarul (`student`) şi grupul (`student`) pot citi şi scrie în fişier. Oricine alt utilizator ce nu aparţine grupului student poate numai să citească fişierul.

Dacă tastăm comanda:

```
chmod o+rw test.txt
```

numai alţii (ce nu sunt nici proprietari şi nici nu aparţin grupului `student`) vor putea să citească şi să scrie fişierul.

[15]Grupul din care face parte proprietarul.

Deoarece utilizatorul student este proprietar al acestui fişier, el va putea întotdeauna să schimbe permisiunile pentru a recăpăta accesul în citire şi scriere.

Comanda ce se va tasta va fi:

```
chmod u+rw test.txt
cat test.txt
acesta este un test
```

S-a verificat şi permisiunea în citire cu ajutorul comenzii `cat`.

Pentru ştergerea tuturor permisiunilor asupra fişierului `test.txt` pentru oricine, se va tasta comanda:

```
chmod a-rw test.txt
```

Să vedem dacă fişierul poate fi citit:

```
cat test.txt
cat: test.txt: Permission denied
```

Mesajul de eroare returnat de sistem, `Permission denied`, semnifică faptul că execuţia comenzii anterioare (`cat`), nu este posibilă datorită lipsei permisiunii de citire necesare.

Alte exemple de setări ce pot fi folosite în cadrul comenzii `chmod` sunt:

`g+w` adaugă permisiunea de scriere pentru grup
`o-rwx` şterge toate permisiunile pentru alţii
`u+x` permite proprietarului fişierului să execute fişierul
`a+rw` pemite oricui să citească şi să scrie în fişier
`ug+r` permite proprietarului şi grupului să citească fişierul
`g=rx` permite grupului numai să citească şi să execute fişierul, fără scriere

O a doua metodă de modificare a permisiunilor asupra fişierelor este cea bazată pe un sistem de codificare numerică. Acesta permite specificarea permisiunilor asupra fişierelor sub forma unui număr din 3 cifre octale.

Este important de a înţelege cum lucrează acest sistemul de codificare, deoarece numerele în cauză sunt folosite, atât pentru a schimba permisiunile asupra fişierelor, cât şi de către mesajele de eroare ce implică permisiunile.

În cadrul unui astfel de număr prima cifră codifică permisiunile pentru proprietar, a doua pe cele pentru grup şi a treia pe cele pentru alţii.

Conform sistemului de codificare respectiv, seturile de câte trei permisiuni (`rwx`) pot fi interpretate ca un număr binar pe 3 biţi. Astfel, o permisiune acordată corespunde unei cifre de 1, iar o permisiune neacordată corespunde unei cifre de 0. Prin urmare pentru setul (`r-x`) combinaţia binară corespondentă va fi `101` care în zecimal este egală cu: `4+0+1=5`.

Cifrele individuale sunt codificate prin însumarea tuturor permisiunilor, "permise" pentru acel utilizator particular[16] după cum urmează:

Permisiune citire	4
Permisiune scriere	2
Permisiune execuţie	1

Sunt posibile următoarele combinaţii:

0 - nici o permisiune
4 - numai citire
2 - numai scriere
1 - execuţie
6 - citire şi scriere
5 - citire şi execuţie
3 - scriere şi execuţie

[16]Deci dintr-un set de trei.

7 - toate

Prin urmare o permisiune asupra unui fişier, codificată prin numărul 751, înseamnă că proprietarul are permisiunile r,w,x (4+2+1=7), grupul are permisiunile r şi x (4+0+1=5) şi alţii au numai permisiunea x (0+0+1=1).

Observaţie:

Setarea permisiunilor 666 sau 777 va permite ca orice utilizator să poată citi şi scrie într-un fişier sau director. Asemenea setări pot determina fraudarea fişierelor, şi ca urmare nu sunt recomandate.

În continuare sunt prezentate câteva dintre cele mai obişnuite setări, valori numerice, precum şi semnificaţia lor:

-rw------- (600) - numai proprietarul are permisiunile r şi w

-rw-r--r-- (644) - numai proprietarul are permisiunile r şi w; grupul şi alţii pot numai să citească

-rwx------ (700) - numai proprietarul are permisiunile r w x

-rwxr-xr-x (755) - proprietarul are permisiunile r w şi x; grupul şi alţii pot numai să citească şi să execute

-rwx--x--x (711) - proprietarul are permisiunile r w x; grupul şi alţii pot numai să execute

-rw-rw-rw- (666) - oricine poate să citească şi să scrie în fişier

-rwxrwxrwx (777) - oricine poate să citească, să scrie, să execute.

Schimbarea permisiunilor asupra directoarelor

Se poate realiza exact în acelaşi fel ca şi în cazul fişierelor, deci tot cu ajutorul comenzii chmod. Orice utilizator care are permisiunea de scriere într-un director poate şterge fişiere din acel director, chiar dacă utilizatorul are sau nu permisiunea de scriere asupra fişierului.

Deoarece nu putem "executa" un director, când se setează sau şterge permisiunea de execuţie asupra acestuia, în fapt se setează sau se şterge permisiunea de a căuta în acel director.

De exemplu să tastăm:

```
chmod a-x aplicatii
```

pentru a şterge permisiunea de execuţie pentru toţi utilizatorii.

Iată ce se întâmplă când se încearcă folosirea comenzii cd în directorul aplicatii:

```
cd aplicatii
bash:aplicatii: Permission denied
```

Să restaurăm permisiunea de execuţie pentru proprietar şi pentru grupul acestuia:

```
chmod ug+x aplicatii
```

Acum, dacă vom verifica cu ajutorul comenzii ls -dl se va vedea că numai others au interzis accesul la directorul aplicatii.

Pentru a permite oricui de a avea acces în citire şi scriere asupra directorului aplicatii se va tasta:

```
chmod -R a+rw aplicatii
```

Prin adăugarea indicatorului -R, se pot schimba permisiunile pentru întreaga structură director.

Observaţii:

1. Modalitatea de a obţine acces la un fişier atunci când nu avem permisiunile necesare, şi fără a ne deconecta de la sistem, este de a utiliza comanda su. Acesta permite conectarea cu numele altui utilizator ce are permisiunile necesare. Evident va trebui să cunoaştem parola utilizatorului respectiv.

2. Unele versiuni de UNIX furnizează un bit suplimentar numit sticky bit ca parte a permisiunii asupra unui director. Scopul acestui bit este de a permite numai proprietarului

directorului, proprietarului fişierului sau utilizatorului root să şteargă şi să redenumească fişiere.

3.4.4. Sistemul de fişiere al UNIX

Un sistem de fişiere reprezintă modul de organizare şi exploatare a informaţiilor stocate pe un suport de memorie externă în vederea accesării şi prelucrării lor de către sistemul de operare.

Din punctul de vedere al utilizatorului, sistemele de fişiere prezintă o organizare bazată pe conceptele de fişier şi director.

Fişierele sunt entităţi care încapsulează informaţia de un anumit tip, iar directoarele grupează în interiorul lor fişiere şi alte directoare.

Deci sistemul de fişiere are o organizare ierarhică, fişierele fiind grupate în directoare care sunt structurate pe mai multe niveluri într-o structură arborescentă. Directorul de nivel cel mai înalt, din cadrul acestei arborescenţe, se numeşte *director rădăcină*, şi este simbolizat prin caracterul /.

Orice fişier sau director poate fi identificat prin numele său indicat ca nume de cale, fie în mod absolut, faţă de directorul rădăcină, fie în mod relativ, faţă de directorul curent.

3.4.4.1 Partiţii

Un sistem de fişiere este în general creat pe hard disc. Sub sistemul de operare UNIX este posibilă instalarea mai multor sisteme de fişiere, fiecare dintre acestea fiind plasat într-o zonă proprie pe hard disc. Aceste zone în care este subdivizat hard discul se numesc partiţii[17].

La nivel utilizator, fiecare partiţie se comportă ca un disc de sine stătător memorând un sistem de fişiere. Informaţia referitoare la partiţii se memorează la începutul discului, în aşa-numita *tabelă de partiţii*. Aceasta conţine 4 intrări (câte una pentru fiecare partiţie posibilă) în care sunt memorate poziţiile, dimensiunile şi tipurile partiţiilor de pe disc. Cele 4 partiţii se numesc partiţii primare, fiind posibil ca în interiorul oricăreia dintre ele să se creeze câte o nouă tabelă de partiţii, referind partiţii care fizic se află în interiorul partiţiei curente şi care se numesc partiţii extinse.

Sistemul de operare UNIX necesită prezenţa a cel puţin o partiţie. Partiţia necesară menţionată anterior se numeşte partiţie root. În cadrul ei rezidă software-ul de sistem şi fişierele de configurare. Deoarece partiţia root nu are menirea de a păstra datele utilizatorilor, este necesară crearea de partiţii separate pentru directoarele home ale utilizatorilor, pentru fişierele temporare, ş.a.

În UNIX, echipamentelor periferice le sunt asociate fişiere de dispozitiv, ceea ce permite utilizarea şi în cazul acestora, a comenzilor ce sunt folosite pentru fişierele obişnuite. Acest lucru este valabil şi pentru hard discuri. Ca urmare, perntru identificarea lor se foloseşte următoarea schemă, ce este funcţie de tipul unităţii de control la care s-a conectat hard discul.

Astfel, sub sistemul Linux, hard discurile conectat la o unitate de control IDE, sunt denumite:

```
/dev/hd[unitate_hard_disc]
```

Fiecare unitate de hard disc IDE este identificată printr-o literă. Astfel, discul master (primul) din cadrul primei unităţi de control este a, cel de-al doilea disc (slave) din cadrul aceleiaşi unităţi de control este b. Primul disc ataşat celei de a doua unităţi de control este c, ş.a.m.d.

Pentru a referi partiţiile create pe astfel de hard discuri, se utilizează numere de ordine. De exemplu, cea de a treia partiţie a unităţii de hard disc slave de pe prima unitatea de

[17]Fiecare partiţie este o mulţime continuă de blocuri. Un bloc este format dintr-un sigur sector (cum este cazul la dischete) sau din mai multe sectoare (cazul hard discurilor).

control este referită: /dev/hdb3.

Unităţile de hard disc conectate unei unităţi de control SCSI folosesc aceeaşi schemă de identificare, cu excepţia că în loc să utilizeze drept prefix construcţia /dev/hd, folosesc /dev/sd, urmată de litera corespunzătoare. Referirea partiţiilor este similară.

Astfel, când ne referim la a doua partiţie de pe primul hard disc ataşat unei unităţi de control SCSI, vom folosi specificaţia: /dev/sda2. Atunci când dorim să ne referim la întregul disc, vom specifica toată informaţia cu excepţia numărului asociat partiţiei. De exemplu, pentru a ne referi la întregul disc master conectat pe prima unitatea de control SCSI, vom utiliza specificaţia: /dev/sda.

În cele două scheme de identificare anterioare, /dev este numele directorului aparţinând arborescenţei standard UNIX, ce conţine fişierele de dispozitiv asociate echipamentelor periferice din cadrul sistemului.

3.4.4.2.Tipuri de fişiere

În cadrul sistemului de fişiere al UNIX există mai multe tipuri de fişiere. Acestea sunt:

- fişiere normale (obişnuite)
- directoare (fişiere catalog)
- legături hard
- legături simbolice
- socket-uri
- conducte cu nume (fişiere FIFO)
- fişiere de dispozitiv (fişiere speciale)

Fişiere normale

Un fişier normal este un fişier permanent, ce este privit de către sistemul de operare ca un şir de octeţi fără o organizare logică specială, şi a cărui structură internă este irelevantă din punctul de vedere al administratorului de sistem.

Un astfel de fişier poate conţine informaţie binară (în cazul unui fişier în format executabil), sau linii de text separate prin caracterul linie nouă (012_8), structurarea sa logică revenind exclusiv în sarcina programatorului.

O comandă ls -l va afişa în cazul unui fişier normal, o informaţie similară cu următoarea:

```
-rw------- 1 student ise 42 May 12 13:09 fistest
```

Se observă că primul caracter afişat este -. Acesta semnifică faptul că fişierul respectiv este un fişier normal.

Directoare

Sunt fişiere de un tip special ce conţin o listă de alte fişiere. Informaţiile din lista respectivă fac referire la numele şi locaţiile fişierelor, mărimea lor, timpul aferent creării şi modificării acestora.

Directoarele pot ele însile să conţină subdirectoare, care la rândul lor pot să conţină alte subdirectoare, ş.a.m.d., formând în acest fel o structură ierarhică (arborescentă).

Se poate aprecia deci că fişierele director reprezintă o modalitate de a structura logic sistemul de fişiere.

Un director poate fi citit numai de către sistemul de operare UNIX sau de către programe special scrise pentru a efectua procesarea acestuia.

Cu ajutorul comenzii ls -l se pot identifica directoarele datorită apariţiei caracterului d ce este plasat înaintea permisiunilor de acces:

```
drwx------ 2 student ise 512 May 12 13:08 public
```

3.4.4.3 Legături la fişiere

În anumite circumstanţe este necesar[18], de a furniza nume alternative pentru un acelaşi fişier.

Acest lucru poate fi realizat prin legarea unui nume de fişier la altul, o legătură fiind o modalitate de a furniza un alt nume unui acelaşi fişier[19]. Legarea unui fişier la un alt nume nu implică duplicarea conţinutului fişierului respectiv.

Este posibil de a lega un fişier la un alt nume în cadrul aceluiaşi director sau la acelaşi nume în cadrul unui alt director.

Orice operaţie care se execută asupra fişierului legătură (mai puţin ştergerea) îşi va avea efectul de fapt asupra fişierului indicat de legătură. Dacă este solicitată ştergerea, efectul depinde de tipul legăturii respective.

Legături hard (`hard links`)

Acest tip de legătură creează o referinţă (un pointer) către un fişier deja existent, fără duplicarea conţinutului fişierului respectiv. Deci, numele original al fişierului şi numele fişierului legătură fac referire către aceeaşi adresă fizică (acelaşi `i-nod`).

Ca urmare, la afişarea conţinutului unui director acesta apare că ar conţine două fişiere identice, lucru ce determină sistemul să le trateze în consecinţă.

Există două limitări importante ale unei legături hard, şi anume: un director nu poate avea o legătură hard, şi o astfel de legătură nu poate exista peste mai multe sisteme de fişiere deoarece ea partajează un `i-nod`.

Este posibil de a şterge numele fişierului original fără a şterge numele fişierului legătură. Sub aceste circumstanţe, fişierul nu este şters, dar intrarea în director a fişierului original va fi ştearsă iar contorul numărului de legături este decrementat cu 1. Blocurile de date ale fişierului original sunt şterse atunci când contorul numărului de legături devine zero.

Legături simbolice (`symbolic links`)

Legăturile simbolice sunt de fapt fişiere distincte, marcate cu un cod special, care au ca şi conţinut numele complet al fişierului indicat.

Există deci două fişiere: unul este fişierul original, şi altul este fişierul legătură ce conţine numele fişierului original.

Spre deosebire de o legătură hard, o legătură simbolică poate să existe peste mai multe sisteme de fişiere şi poate fi folosită pentru a lega atât directoare cât şi fişiere.

Dezavantajul unui legătrui simbolice este că pentru ea (fiind fişier) trebuie creat un i-nod separat şi, în plus, ocupă spaţiu pe disc prin conţinutul ei.

Ştergerea unei astfel de legături nu afectează fişierul original, dar dacă acesta este şters, legătura simbolică va face referire către un fişier care nu mai există.

Pentru a genera legături hard sau simbolice la fişiere, se utilizează o comandă specifică, numită `ln`. Sintaxa ei generală este:

```
ln nume_fisier nume_legatura
```

unde `nume_fisier` este fişierul ce există deja, iar `nume_legatura`, este numele fişierului legătură creat.

Implicit comanda creează o legătură hard.

De exemplu, dacă în directorul curent există un fişier numit `test1`, şi tastăm comanda `ls -l`, se vor afişa următoarele informaţii:

```
-rw------- 1 stud1 ise 42 May 12 13:04 test1
```

Pentru a lega hard fişierul `test1` la fişierul `test2` în directorul curent, vom executa comanda:

```
ln test1 test2
```

[18]De exemplu, dacă dorim să permitem accesul altor utilizatori, la propriile noastre fişiere, fără ca pentru aceasta să le punem la dispoziţie o copie a fişierelor respective.

[19]Practic, o legătură este văzută de către utilizator ca un fişier cu un nume propriu, dar care în realitate face referire la un alt fişier de pe disc.

Tastând din nou comanda `ls -l`, se vor afişa informaţiile:

```
-rw-------  2 stud1 ise 42 May 12 13:04 test2
-rw-------  2 stud1 ise 42 May 12 13:04 test1
```

Apar deci două fişiere care au aceeaşi mărime, 42. Totodată numărul de legături (coloana a doua), a fost incrementat de la 1 la 2.

Tastând comanda `ls -il`, se vor afişa informaţiile:

```
9180 -rw------- 2 stud1 ise 42 May 2 8:04 test2
9180 -rw------- 2 stud1 ise 42 May 2 8:04 test1
```

Se observă că ambele nume de fişier fac referire către un acelaşi `i-nod, 9180`. Este deci vorba de două nume care fac referire către un acelaşi fişier.

Unul dintre indicatorii cei mai utilizaţi ai comenzii `ln` este `-s`, folosirea acestuia pemiţând generarea unei legături simbolice. Formatul comenzii `ln`, în acest caz, este:

```
ln -s nume_fisier nume_legatura
```

De exemplu, pentru a crea o legătură simbolică la fişierul `test1` în directorul curent, se va tasta comanda următoare:

```
ln -s test1 test2
```

Aceasta creează un fişier `test2` legat, care va conţine numele lui `test1.`

Tastând o comandă `ls -l` în directorul curent, se vor afişa informaţiile:

```
-rw-------  2 stud1 ise 42 May 12 13:04 test1
lrw------- 1 stud1 ise 42 May 12 13:04 test2→test1
```

Fişierul `test2` apare marcat ca fiind o legătură simbolică în cadrul sistemului de fişiere.

Socket-uri

Un `socket` nu este în fapt un fişier real, ci un mecanism folosit de `UNIX` pentru a comunica între două calculatoare gazdă, utilizând pentru aceasta porturi reţea.

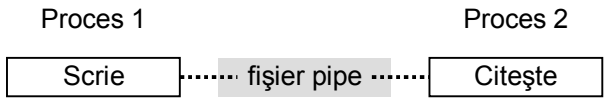
Fişierele `socket` sunt identificate de către setările lor de permisiuni ce încep cu caracterul `s.` O comandă `ls -l` asupra unui fişier `socket` afişează o informaţie similară cu următoarea:

```
srwxrwxrwx 1 root admin 0 May 10 14:38 X0
```

Pentru ştergerea unui fişier `socket` se va folosi comanda `rm.`

Conducte cu nume (`named pipes`)

O conductă cu nume este un fişier temporar ce permite comunicaţia între procese prin mecanismul `pipe` (conductă). Astfel, procesul emitent scrie date în conducta cu nume, iar procesul receptor citeşte datele din aceasta. Procesarea datelor din conducta cu nume se face pe baze FIFO.



Existenţa unui astfel de fişier este limitată la intervalul de timp în care procesele comunică.

Recunoaşterea conductelor cu nume se face datorită caracterului `p` cu care începe informaţia aferentă permisiunilor lor de acces, informaţie afişată la executarea unei comenzi `ls - l:`

```
prw------- 1 root admin 0 May 12 22:02 nume_conducta
```

Fişiere de dispozitiv

O particularitate care diferenţiază sistemul de operare `UNIX` de alte sisteme, o constituie asocierea dispozitivelor periferice cu fişiere de dispozitiv (fişiere speciale). Fişierele de dispozitiv sunt citite/scrise, din punctul de vedere al utilizatorului, exact ca şi cele normale, rezultatul unei astfel de operaţii fiind activarea driver-ului asociat perifericului respectiv. Un

program de aplicaţie poate deci utiliza aceeaşi sintaxă pentru a accesa un fişier normal sau un fişier de dispozitiv. Fie, de exemplu, comanda de copiere a fişierului `test.p:`

```
cp test.p /usr/diploma/copie.p
```

Prin lansarea acestei comenzi se realizează copierea fişierului `test.p` în fişierul destinaţie `/usr/diploma/copie.p`. Dacă numele fişierului destinaţie se înlocuieşte cu un nume de fişier de dispozitiv, de exmplu imprimanta (`/dev/lp0`), obţinem o listare a fişierului `test.p`.

```
cp test.p /dev/lp0
```

Unui dispozitiv periferic îi este asociat cel puţin un fişier de dispozitiv, care va conţine întotdeauna informaţii despre driver-ul (programul de comandă) al acelui periferic. Toate fişierele de dispozitiv rezidă în directorul `/dev`.

Spre exemplu, fiecărei unităţi de disc îi corespunde câte un fişier în directorul `/dev`. În `Linux`, primei unităţi de dischetă îi corespunde fişierul `/dev/fd0`, celei de-a doua `/dev/fd1`, ş.a.m.d. Primei unităţi de hard disc cu interfaţă `IDE`, din sistem, îi corespunde fişierul special `/dev/hda`, iar primei sale partiţii, fişierul `/dev/hda1`. A doua partiţie de pe primul disc are ca şi corespondent fişierul `/dev/hda2`, a doua unitate de hard disc `IDE` este referită ca fiind `/dev/hdb`, ş.a.m.d.

În funcţie de unitatea de informaţie pe care o transmit/recepţionează la un moment dat, dispozitivele periferice se împart în două tipuri:

- dispozitive orientate pe caractere (terminale, imprimante)
- dispozitive orientate pe blocuri (unităţi de disc)

Evident, funcţie de această caracteristică, dispozitivelor în cauză le corespund fişiere de dispozitiv specifice.

În cadrul unui sistem pot exista mai multe dispozitive periferice de acelaşi tip. Pentru a distinge între ele, `UNIX`-ul asociază fiecărui dispozitiv, două numere:

- un număr major, care indică tipul dispozitivului (de exemplu, imprimanta)
- un număr minor, care indică al câtelea dispozitiv periferic este în clasa de dispozitive de acelaşi tip (de exemplu, a doua imprimantă)

Fişierele de dispozitiv caracter se găsesc localizate în directorul `/dev` şi furnizează un mecansim pentru comunicarea, un caracter la un moment dat, cu drivere-le dispozitivelor din sistem.

Informaţia afişată de o comandă `ls -l` în cazul unui astfel de fişier, este similară cu următoarea:

```
crw-rw-rw- 1 root wheel 21, 4 May 12 13:40 ptyp4
```

Se observă că permisiunile de acces în acest caz încep cu caracterul `c`.

Fişierele de dispozitiv bloc se găsesc, de asemenea, localizate în directorul `/dev`, fiind folosite pentru a comunica cu drivere-le de dispozitiv. Ele sunt asociate însă, unor dispozitive periferice ce transferă blocuri mari de date la un moment dat (cum este hard discul).

Aceste fişiere sunt identificate de către caracterul b ce apare în faţa permisiunilor de acces, atunci când se tastează comanda `ls -l:`

```
brw-------  2 root staff 16, 2 Jul 29 2006 fd0c
```

Crearea fişierelor de dispozitiv, bloc sau caracter, cât şi a conductelor cu nume, se poate realiza utilizând comanda `mknod`. Sintaxa acestei comenzi este:

`mknod fisier_dispozitiv b|c|p|u major minor,` unde:

`b` semnifică faptul că `fisier_dispozitiv` este un fişier asociat unui dispozitiv bloc

`c` şi `u` semnifică faptul că `fisier_dispozitiv` este un fişier asociat unui dispozitiv caracter

`p` semnifică faptul că `fisier_dispozitiv` este un fişier asociat unei conducte cu nume

Unul dintre aceste argumente trebuie să fie prezent în sintaxa comenzii. Este necesară, de asemenea, specificarea numerelor major şi minor (cu excepţia cazului când se creează o conductă cu nume). Ştergerea fişierelor create cu comanda `mknod` se face cu ajutorul comenzii `rm`.

CAP.4. PROIECTAREA SISTEMELOR INFORMATICE

4.1. SISTEME INFORMATICE

Din punct de vedere funcţional, sistemul informatic are mai multe definiţii, în funcţie de scop, elementele din structură şi relaţiile dintre elemente; cea mai generală definiţie îl prezintă ca pe *un sistem de colectare, memorare, prelucrare şi distribuire a informaţiilor care utilizează calculatorul electronic*.

Fiecărui sistem informatic i se asociază un sistem de prelucrare a datelor, în care datele se prezintă pe diferiţi suporţi de memorare, iar procesele de prelucrare sunt concretizate în proceduri (automate şi manuale) executate de diferite echipamente de tehnică de calcul şi teletransmisie şi de către personal specializat.

Punând în evidenţă faptul că utilizarea tehnicii de calcul a produs modificări majore în modul de realizare a activităţilor informaţionale, Gh. Sabău şi I. Lungu definesc sistemul informatic ca fiind *un ansamblu de elemente interconectate funcţional în scopul automatizării obţinerii informaţiei şi fundamentării deciziilor*[1].

Există alte opinii potrivit cărora sistemul informatic poate fi avut în vedere atât ca model extern, cât şi sub aspectul unui model intern[2].

Modelul extern al sistemului informatic se referă la arhitectura funcţională orientată către utilizator, pe care îl interesează informaţiile şi procesele lor de prelucrare.

Modelul intern al sistemului informatic se referă la structura fizică orientată către echipamente, unde interesează datele, procedurile de prelucrare, echipamentele, mediile de programare şi sistemele de operare.

Tipuri de sisteme informatice. Sistemele informatice pot fi clasificate în funcţie de diferite criterii[3].

I. Clasificarea în concordanţă cu nivelurile organizatorice. Organizaţiile sunt compuse din componente cum ar fi departamente, divizii, echipe etc. Aceste departamente raportează unui nivel organizatoric mai înalt cum ar fi o divizie sau un sediu central într-o manieră ierarhică. Aceasta este o structură ierarhică tradiţională cu mai multe niveluri.

O cale pentru organizarea sistemului informatic este construirea lui în conformitate cu structura organizatorică. Aceste sisteme pot fi independente sau pot fi interconectate. De exemplu, un departament de resurse umane poate fi atât la nivel central cât şi cât şi în fiecare divizie sau sucursală. Sistemele informatice construite conform cu nivelurile organizatorice sunt:

1. Sisteme informatice departamentale. Frecvent, o organizaţie foloseşte mai multe programe aplicative într-o singură arie funcţională. De exemplu în gestiunea personalului, este posibil să se folosească un program pentru selectarea solicitanţilor de locuri de muncă şi un program pentru monitorizarea fluctuaţiei personalului. Unele aplicaţii sunt complet independente în timp ce altele sunt interconectate. Colecţia de programe aplicative în domeniul resurselor umane se poate numi *Sistem informatic pentru resurse umane.*

2. Sisteme informatice organizaţionale. În timp ce un sistem informatic departamental este în mod obişnuit legat de o arie funcţională, se poate vorbi frecvent de colecţii de aplicaţii în mai multe sau în toate ariile funcţionale. O astfel de colecţie poate fi descrisă ca un sistem informatic organizaţional.

3. Sisteme informatice interorganizaţionale. Unele sisteme informatice sunt foarte complexe şi implică mai multe organizaţii. De exemplu sistemul de rezervare a biletelor de avion în lumea întreagă este compus din mai multe sisteme aparţinând diferitelor companii aeriene.

---

[1] Sabău, Gh., Lungu, I. - *Sisteme informatice şi baze de date,* ASE, Bucureşti, 1993, p. 229

[2] Turbuţ, Gh. (coordonator) - Inginerie de sistem, automatizări şi informatică în transporturi, vol. 2, Editura Tehnică, Bucureşti, 1989

[3] Turban, E., McLean, E., Wetherbe, J., - Information Tehnology for Management, John Wiley & Sons, New-York, 1996

Din punct de vedere funcţional, sistemul informatic are mai multe definiţii, în funcţie de scop, elementele din structură şi relaţiile dintre elemente; cea mai generală definiţie îl prezintă ca pe *un sistem de colectare, memorare, prelucrare şi distribuire a informaţiilor care utilizează calculatorul electronic*.

Fiecărui sistem informatic i se asociază un sistem de prelucrare a datelor, în care datele se prezintă pe diferiţi suporţi de memorare, iar procesele de prelucrare sunt concretizate în proceduri (automate şi manuale) executate de diferite echipamente de tehnică de calcul şi teletransmisie şi de către personal specializat.

Punând în evidenţă faptul că utilizarea tehnicii de calcul a produs modificări majore în modul de realizare a activităţilor informaţionale, Gh. Sabău şi I. Lungu definesc sistemul informatic ca fiind *un ansamblu de elemente interconectate funcţional în scopul automatizării obţinerii informaţiei şi fundamentării deciziilor*[20].

Aceiaşi autori identifică în structura sistemului informatic următoarele elemente:

1. Baza tehnică (hardware-ul sistemului informatic), cuprinde totalitatea mijloacelor tehnice de culegere, transmitere, prelucrare şi stocare a datelor.

2. Sistemul de programe (software-ul), se referă la totalitatea programelor necesare funcţionării sistemului informatic în conformitate cu funcţiile şi obiectele ce i-au fost stabilite, respectiv programele de bază şi programele aplicative.

3. Baza ştiinţifico-metodologică este constituită din sistemul indicatorilor economici, procese şi fenomene economice, precum şi din metodologiile de realizare a sistemelor informatice.

4. Baza informaţională cuprinde datele supuse prelucrării fluxurilor informaţionale, sistemele şi nomenclatoarele de coduri.

5. În **resursele umane** se include personalul implicat cu funcţionarea sistemului informatic, iar **cadrul organizatoric** este cel specificat în regulamentul de organizare şi funcţionare al organismului economic în care funcţionează sistemul informatic.

Sistemul informatic al unei întreprinderi este definit de Robert Reix ca fiind *elementul care înglobează toate componentele automatizate ale căror interacţiuni sunt la nivel informaţional. Obiectivul său constă în furnizarea la diferite niveluri de organizare a informaţiilor ce pot însoţi şi controla funcţionarea întreprinderii*[21].

Există alte opinii potrivit cărora sistemul informatic poate fi avut în vedere atât ca model extern, cât şi sub aspectul unui model intern[22].

Modelul extern al sistemului informatic se referă la arhitectura funcţională orientată către utilizator, pe care îl interesează informaţiile şi procesele lor de prelucrare.

Modelul intern al sistemului informatic se referă la structura fizică orientată către echipamente, unde interesează datele, procedurile de prelucrare, echipamentele, mediile de programare şi sistemele de operare.

Tipuri de sisteme informatice. Sistemele informatice pot fi clasificate în funcţie de diferite criterii[23].

I. Clasificarea în concordanţă cu nivelurile organizatorice. Organizaţiile sunt compuse din componente cum ar fi departamente, divizii, echipe etc. Aceste departamente raportează unui nivel organizatoric mai înalt cum ar fi o divizie sau un sediu central într-o manieră ierarhică. Aceasta este o structură ierarhică tradiţională cu mai multe niveluri.

O cale pentru organizarea sistemului informatic este construirea lui în conformitate cu structura organizatorică. Aceste sisteme pot fi independente sau pot fi interconectate. De exemplu, un departament de resurse umane poate fi atât la nivel central cât şi cât şi în fiecare

---

[20] Sabău, Gh., Lungu, I. - *Sisteme informatice şi baze de date,* ASE, Bucureşti, 1993, p. 229

[21] Reix, R. - Systemes d'information et management des organisation, Vuibert, Paris, 1995, p.46;

[22] Turbuţ, Gh. (coordonator) - Inginerie de sistem, automatizări şi informatică în transporturi, vol. 2, Editura Tehnică, Bucureşti, 1989

[23] Turban, E., McLean, E., Wetherbe, J., - Information Tehnology for Management, John Wiley & Sons, New-York, 1996

divizie sau sucursală. Sistemele informatice construite conform cu nivelurile organizatorice sunt:

1. Sisteme informatice departamentale. Frecvent, o organizaţie foloseşte mai multe programe aplicative într-o singură arie funcţională. De exemplu în gestiunea personalului, este posibil să se folosească un program pentru selectarea solicitanţilor de locuri de muncă şi un program pentru monitorizarea fluctuaţiei personalului. Unele aplicaţii sunt complet independente în timp ce altele sunt interconectate. Colecţia de programe aplicative în domeniul resurselor umane se poate numi *Sistem informatic pentru resurse umane.*

2. Sisteme informatice organizaţionale. În timp ce un sistem informatic departamental este în mod obişnuit legat de o arie funcţională, se poate vorbi frecvent de colecţii de aplicaţii în mai multe sau în toate ariile funcţionale. O astfel de colecţie poate fi descrisă ca un sistem informatic organizaţional.

3. Sisteme informatice interorganizaţionale. Unele sisteme informatice sunt foarte complexe şi implică mai multe organizaţii. De exemplu sistemul de rezervare a biletelor de avion în lumea întreagă este compus din mai multe sisteme aparţinând diferitelor companii aeriene. Deci sistemele informatice interorganizaţionale sunt formate din mai multe sisteme informatice organizaţionale la care se adaugă sistemele de legătură dintre acestea.

II. Clasificarea în concordanţă cu ariile funcţionale. După cum s-a observat, sistemele informatice departamentale sunt construite în concordanţă cu ariile funcţionale tradiţionale:

1. Sistemul financiar-contabil,
2. Sistemul pentru producţie,
3. Sistemul pentru activitatea comercială,
4. Sistemul pentru gestiunea resurselor umane,
5. Sistemul pentru management.

În fiecare arie funcţională, sunt efectuate operaţii de bază computerizate care sunt esenţiale pentru activitatea organizaţiei. Pregătirea unui ştat de plată sau înregistrarea unui client sunt exemple tipice. Asemenea operaţii sunt numite *operaţii scop central* (mission-central tasks). Sistemele informatice care suportă asemenea operaţii se numesc sisteme tranzacţionale (TPS). Sistemele tranzacţionale execută operaţii care se efectuează în toate ariile funcţionale dar în special în contabilitate şi finanţe.

III. Clasificarea în concordanţă cu suportul furnizat de sistem. A treia modalitate de a clasifica sistemele informatice este in concordanţă cu tipul suportului pe care îl furnizează, indiferent de aria funcţională implicată. De exemplu, un sistem informatic poate fi un suport pentru funcţionari în aproape orice arie funcţională. Similar, managerii, indiferent unde lucrează, pot utiliza un sistem computerizat ca suport pentru luarea deciziilor.

Principalele sisteme după această clasificare sunt:

1. Sisteme tranzacţionale (TPS - Transaction Processing System);
2. Sisteme manageriale (MIS - Management Information System);
3. Sisteme automatizate pentru birouri (OAS - Office Automation System);
4. Sisteme suport pentru grup (GSS - Group Support System);
5. Sisteme suport pentru decizii (DSS - Decision Support System);
6. Sisteme executive sau sisteme suport (EIS - Executive Information System or Support System), utilizate în activitatea de execuţie;
7. Sisteme suport inteligente (ISS - Intelligent Support System), cuprind sisteme expert (ES - Expert Systems) şi reţele neuronale artificiale (ANN - Artificial Neural Networks).

Implementarea unei aplicaţii specifice poate implica două sau mai multe din sistemele prezentate mai sus. De exemplu, un sistem suport pentru decizii poate fi combinat cu un sistem expert pentru a realiza un sistem informatic care să fie utilizat în realizarea unui program promoţional de marketing.

IV. Clasificarea în concordanţă cu arhitectura sistemelor informatice - depinde de scopul pentru care a fost construit. Arhitectura şi infrastructura sunt două aspecte ale sistemelor informatice ce se intercondiţionează reciproc.

Sistemele informatice pot fi clasificate în concordanţă cu arhitectura lor şi principalele

categorii sunt:

1. Sisteme informatice bazate pe mainframe-uri;
2. Sisteme informatice bazate pe PC-uri;
3. Sisteme informatice distribuite.

V. Clasificarea după natura activităţilor pentru care sunt construite sistemele informatice. Potrivit acestei clasificări sistemele informatice se împart în:

1. sisteme operaţionale;
2. sisteme tactice (manageriale);
3. sisteme strategice.

Sistemele operaţionale oferă suportul necesar operaţiilor zilnice ale unei organizaţii cum ar fi înregistrarea unei facturi sau a numărului de ore pe care le lucrează un muncitor. De asemenea, acestea oferă suport şi deciziilor operaţionale, decizii care se referă la un interval de timp scurt. Sistemele informatice necesare activităţilor şi deciziilor operaţionale sunt în general cele tranzacţionale, executive precum şi sistemele informatice suport pentru decizii. Sistemele operaţionale sunt utilizate de către conducătorii de la nivelele joase, operatori şi funcţionari.

Sisteme tactice (numite şi **sistemele manageriale)** sunt legate de activităţile efectuate de către managerii de la nivelele medii ale întreprinderii, activităţi precum planificarea organizarea şi controlul pe termen scurt şi mediu. Sistemele tactice faţă de cele operaţionale au o sferă mai largă de cuprindere. Sistemele manageriale se bazează în general pe sistemele informatice manageriale, dar şi pe sistemele informatice suport pentru decizii, sisteme informatice executive şi chiar sisteme informatice inteligente.

Sistemele strategice sunt legate de situaţiile pe termen lung şi de deciziile care schimbă semnificativ maniera în care vor fi realizate obiectivele firmei. În mod tradiţional sistemele strategice implică doar planificarea de lungă durată. Cele mai utilizate sisteme informatice ca suport pentru sistemele strategice sunt cele suport pentru luarea deciziilor şi sistemele expert.

4.2. METODOLOGII DE REALIZARE A SISTEMELOR INFORMATICE

Realizarea sistemelor informatice este o activitate complexă care îmbină un mare **număr de activităţi eterogene**: analiză, proiectare, programare cu un pronunţat caracter creativ şi la care cooperează mai multe unităţi organizatorice, ocazionând eforturi materiale, financiare şi umane pe o perioadă îndelungată.

Folosirea eficientă a resurselor şi necesitatea realizării unor sisteme informatice performante au repus ordonarea într-o succesiune bine stabilită pe etape şi subetape a acestui proces determinând conturarea unor metodologii de realizare a sistemelor informatice.

Prin metodologia de realizare a sistemelor informatice se stabilesc: **componentele procesului de realizare**:

- etapele, subetapele, activităţile şi conţinutul lor;
- fluxul executării componentelor;
- metodele, tehnicile şi instrumentele de realizare.

4.2.1. Etapele de realizare a sistemelor informatice

Sistemul informatic are un **ciclu propriu de viaţă**, care începe cu decizia de realizare, cuprinde faza de elaborare, faza de utilizare, faza de perfecţionare şi se încheie cu decizia de abandonare în forma existentă şi înlocuirea cu un nou sistem.

Acestui ciclu de viaţă îi corespund etape specifice stărilor succesive prin care trece sistemul informatic, etape caracterizate prin activităţi distincte. **Etapele realizării unui sistem informatic sunt**:

- analiza sistemului informaţional existent (analiza de sistem);
- proiectarea sistemului informatic;
- elaborarea şi testarea programelor;

- implementarea sistemului informatic;
- exploatarea curentă şi menţinerea în funcţiune a sistemului informatic.

1) Analiza sistemului informaţional existent urmăreşte delimitarea ariei de cuprindere a sistemului şi formularea cerinţelor şi restricţiilor globale de realizare. Pentru a atinge acest scop, în această etapă se face un studiu amănunţit al sistemului existent, se apreciază măsura în care sistemul existent este capabil să răspundă în continuare exigenţelor conducerii ştiinţifice a agentului economic, se apreciază oportunitatea realizării unui sistem informatic şi se formulează principalele restricţii şi cerinţe pentru viitorul sistem informatic.

2) Proiectarea constă în definirea modelului de ansamblu (conceptual) al sistemului informatic, ţinând seama de evaluările făcute în etapa anterioară, dar şi în transformarea modelului conceptual stabilit anterior într-un model tehnic, operaţional.

În acest scop se proiectează ieşirile noului sistem, se determină entităţile bazei informaţionale de intrare, se codifică atributele şi se proiectează fluxul general de prelucrare a datelor în noul sistem. De asemenea, se definitivează soluţia în organizarea datelor (fişiere sau baze de date), se proiectează fişierele sau bazele de date, se determină şi se proiectează procedurile (programele de aplicaţie) pentru crearea, actualizarea şi exploatarea structurilor de date în vederea realizării obiectivelor stabilite sistemului informatic.

3) Elaborarea programelor are la bază soluţiile stabilite în proiectare şi urmăreşte scrierea şi testarea individuală a programelor utilizând medii şi tehnici de programare adecvate.

4) Implementarea sistemului informatic realizat constă în verificarea modului de comportare practică a modelului proiectat şi realizat în vederea trecerii lui în exploatare curentă.

5) Exploatarea curentă şi menţinerea în funcţiune urmăreşte atât îndeplinirea obiectivelor iniţiale ale sistemului informatic cât şi adaptarea acestuia la modificările intervenite în cerinţele informaţionale ale beneficiarului.

Realizarea unui sistem informatic se concretizează şi sub forma unui proiect de sistem informatic prin intermediul căruia se definesc într-o formă standardizată soluţiile adoptate.

4.2.2. Ordinea executării etapelor

Dacă etapizarea procesului de realizare a sistemelor informatice, deşi controversată, este în general stabilită, cu privire la ordinea parcurgerii etapelor există diferite orientări generate de diversitatea sistemelor informatice şi a realizatorilor lor.

Aceste orientări sunt concretizate în:

- modelul liniar;
- modelul cu prototip;
- modelul cu extensii;
- modelul documentaţiei anticipate;
- modelul ierarhic;
- modele mixte.

Modelul liniar este un model teoretic şi presupune parcurgerea secvenţială a etapelor cu eventuale reveniri doar la etapa precedentă. Este aplicat doar pentru sisteme informatice de mică anvergură, dar în realitate parcurgerea etapelor este un proces iterativ, desfăşurându-se în paralel cu mai multe activităţi din etape diferite de realizare.

Modelul cu prototip se aplică atunci când se ia decizia elaborării complete, rapide şi la costuri scăzute a unei versiuni iniţiale, simplificate cu caracter de prototip pe baza căreia se stabilesc noi specificaţii de definire a sistemului informatic şi se desfăşoară activităţi de realizare a unei noi versiuni a sistemului informatic. Elaborarea noii versiuni se poate face prin parcurgerea integrală sau parţială a etapelor şi se modifică numai anumite părţi din prototip.

Modelul cu extensii porneşte de la premisa că sistemele informatice se pot realiza şi da în funcţiune parţial pe aplicaţii. Realizarea lor se face în manieră extensibilă astfel încât o primă versiune să includă componentele de bază, celelalte urmând să fie realizate şi integrate prin

extensii ulterioare. Se realizează astfel reducerea eforturilor şi fructificarea experienţei câştigate în elaborarea componentelor iniţiale, deci rezultă că extensiile se ramifică din proiectarea generală.

Modelul documentaţiei anticipate presupune întocmirea în devans a documentaţiei prin parcurgerea etapelor de proiectare, deci el permite analiza critică a documentaţiei urmată de reformularea unor probleme şi aspecte înainte de darea în exploatare a sistemului.

Modelul ierarhic se caracterizează prin structurarea ierarhică coerentă a activităţilor corespunzătoare fiecărei etape, conţinutul acestora fiind conform specificului fiecărui nivel.

Modele mixte. În activitatea practică se întâlnesc combinaţii ale modelelor de parcurgere a etapelor anterior prezentate, adaptate condiţiilor concrete de realizare a sistemelor informatice.

O metodologie este o implementare a ciclului de viaţă al sistemului informatic care face referire la:

- activităţile din fiecare etapă şi fază şi ordinea lor de desfăşurare;
- rolul individual şi de grup pe care îl are fiecare activitate;
- condiţiile de livrare şi standardele de calitate pentru fiecare activitate;
- instrumentele şi tehnicile care vor fi utilizate de fiecare activitate.

Există mai multe metodologii de realizare a sistemelor informatice, din care cele mai frecvent utilizate la noi în ţară sunt:

A. SSADM, care se bazează pe analiza structurată şi pune accentul pe prelucrări;
B. MERISE;
C. OMT, care integrează datele şi prelucrările pentru a crea obiecte care pot fi uşor adaptate şi reutilizate;
D. Prototipizarea, care pune accentul pe realizarea şi asamblarea de prototipuri.

A. Elemente de bază ale metodologiei SSADM.

SSADM (Structured System Analysis and Design Methology) este o metodologie structurată de analiză şi proiectare a sistemelor informatice. A fost dezvoltată în Marea Britanie, şi se bazează pe trei principii metodologice după cum urmează:

- folosirea modelării descendente, de tip top-down, ca mod de abordare; acest mod de abordare permite să se pornească de la cerinţele conducerii şi să se detalieze cerinţele până la nivelul utilizatorilor finali;
- considerarea datelor şi prelucrărilor egale ca importanţă;
- folosirea unor simboluri grafice unice pe toată durata de realizare a sistemului.

Metodologia SSADM se concentrează cu precădere pe acele etape din ciclul de viaţă care se referă la elaborarea sistemului: analiza sistemului existent; proiectarea logică; proiectarea fizică.

Tehnicile de analiză şi proiectare specifice acestei metodologii sunt: diagrama de flux a datelor; modelul entităţilor; matricea entităţi-funcţiuni; normalizarea; structurarea ierarhică a prelucrărilor; documentarea.

Dacă se utilizează această metodologie, echipa de realizare trebuie să acorde atenţie deosebită fiecărei faze şi să întocmească documentaţia pentru fiecare etapă, fapt care conduce la micşorarea duratei de realizare şi a costului proiectului.

B. Elemente ale metodologiei MERISE.

Această metodologie elaborată în Franţa de Ministerul Industriei. Are la bază un model tridimensional care structurează componentele metodologice pe trei cicluri:

- ciclul de abstractizare;
- ciclul de decizie;
- ciclul de viaţă al sistemului informatic.

Ciclul de abstractizare se dezvoltă, conform modelului ANSI/SPARC de definire a bazelor de date, pe trei niveluri:

- nivelul conceptual,
- nivelul logic,
- nivelul fizic,

cu precizarea că modelul datelor se dezvoltă independent de modelul prelucrărilor.

În cadrul acestui ciclu se realizează succesiv modelul conceptual al datelor şi modelul conceptual al prelucrărilor, apoi se elaborează modelul logic al datelor şi modelul logic al prelucrărilor (sau modelul organizaţional) şi apoi se dezvoltă modelul fizic al datelor şi modelul fizic al prelucrărilor (sau modelul operaţional).

Ciclul de decizie se compune din mulţimea deciziilor ce se iau pe durata ciclului de viaţă al sistemului informatic, aceste decizii fiind structurate pe tipuri de decizii astfel:

- decizii generale,
- decizii organizatorice,
- decizii tehnice,
- decizii funcţionale.

C. Elemente ale metodologiei OMT (Object Modeling Technique)

Creată de către James Rumbaugh, este cea mai cunoscută şi utilizată metodologie de proiectare orientată pe obiecte (POO). Dorinţa de a construi produse software prin asamblarea de componente, adică în aceeaşi manieră în care se construieşte hardware-ul, a stat la baza metodelor POO. Astfel, un sistem informatic este privit ca un ansamblu de obiecte care cooperează între ele, iar obiectele sunt tratate ca instanţe ale unei clase în interiorul unei ierarhii. Orientarea pe obiecte forţează ca toate structurile de date ale unui program să fie încapsulate.

Etapele parcurse în cadrul metodologiei OMT sunt:

1. definirea problemei,
2. elaborarea unei modalităţi informale de identificare a datelor şi funcţiilor relevante din cadrul problemei,
3. formalizarea strategiei prin:
    a) identificarea obiectelor şi a atributelor lor,
    b) identificarea operaţiilor asupra obiectelor,
    c) stabilirea instanţelor,
    d) implementarea prelucrărilor.

Metodologia OMT foloseşte pentru proiectarea sistemelor informatice ca mijloace de reprezentare diagrama de flux de date pentru surprinderea comportamentului dinamic şi diagrama modulară pentru surprinderea comportamentului static.

Printre avantajele acestei metodologii se pot enumera:

- produsul informatic este construit din componente care au o calitate probată în implementările anterioare;
- un obiect poate fi modificat fără a afecta celelalte obiecte şi deci sistemul informatic astfel construit este modular şi poate fi uşor dezvoltat şi modificat;
- prin reutilizarea componentelor existente se asigură un timp mai scurt şi costuri mai reduse în procesul de realizare a sistemului informatic.

4.2.3. Metode şi tehnici de realizare a sistemelor informatice

În activitatea practică au apărut şi s-au dezvoltat două tipuri de strategii de abordare şi realizare a sistemelor informatice: în funcţie de rolul sistemelor informatice în cadrul agentului economic şi în funcţie de modul de abordare a subsistemelor si aplicaţiilor ce compun sistemul informatic.

A. În ceea ce priveşte strategia de realizare a unui sistem informatic, în funcţie de **rolul sistemelor informatice** în cadrul agentului economic rezultă **trei categorii de strategii**: ameliorative; inovatoare; adaptive.

Strategiile **ameliorative** urmăresc automatizarea activităţilor şi operaţiilor de rutină sau cu caracter repetitiv. Sistemele informatice ce rezultă nu antrenează mari modificări în sistemul agentului economic. Ele au grad redus de complexitate, nu folosesc personal numeros, timpul de realizare şi implementare este scurt, dar sistemul este lipsit de flexibilitate.

În cazul strategiilor **inovatoare**, introducerea sistemului informatic trebuie însoţită de schimbări importante în organizarea şi funcţionarea agentului economic. Aceste strategii asigură valorificarea superioară a calculatorului electronic, dar presupun un timp mai mare pentru conceperea şi realizarea noului sistem, cheltuieli sporite şi personal de înaltă calificare.

În cazul strategiilor **adaptive**, sistemul informatic este conceput şi realizat astfel încât să fie compatibil cu schimbările care apar în cerinţele informaţionale şi organizatorice şi în funcţionarea agentului economic. La baza acestor strategii stă principiul invariaţiei sau modificării lente impuse de existenţa proceselor şi structurii de bază ale agentului economic. Elementele informaţionale invariante sunt surprinse în baza informaţională care ocupă locul central în sistemul informatic. Aceste strategii au avantajul unei flexibilităţi ridicate datorită conceptului de bază informaţională.

Prin comparaţie, strategiile ameliorative pleacă de la un set bine precizat de ieşiri în funcţie de care se determină celelalte elemente ale sistemului informatic: bazele de date, algoritmi, proceduri, iar strategiile adaptive pleacă de la determinarea setului de intrări strict necesare asigurării bazei informaţionale astfel încât acestea să modeleze cât mai fidel sistemul agentului economic.

Baza informaţională se transpune în colecţii de date, proceduri de creare şi actualizare care constituie nucleul sistemului informatic. Acesta dă posibilitatea modificării cerinţelor informaţionale astfel încât să se asigure reglarea fenomenelor şi proceselor economice.

B. În funcţie de **modul de abordare** a subsistemelor si aplicaţiilor ce compun sistemul informatic pot exista **trei tipuri de strategii**: descendentă, ascendentă şi mixtă.

Metoda descendentă TOP-DOWN este impusă de faptul că proiectarea sistemelor informatice de mare complexitate face necesară descompunerea ierarhică prin modularizare ca modalitate de control a complexităţii. Ea constă în descompunerea unui sistem complex pe niveluri ierarhice, succesiv, până la module elementare, simple şi relativ independente care sunt controlate de module coordonatoare.

Metoda are ca cerinţă de aplicare modularizarea sistemului, deci obiectivul principal este realizarea modularizării de sus în jos, iar din obiectivul principal rezultă următoarele obiective specifice: crearea posibilităţii de realizare în paralel a componentelor sistemului informatic şi eliminarea din sistem a redundanţelor.

Modul de lucru în cadrul acestei metode este următorul :

- se realizează mai întâi o descompunere a sistemului în funcţii principale şi se stabilesc relaţiile dintre acestea, aceste funcţii stând la baza unei prime modularizări;
- se face apoi analiza modulelor obţinute şi dacă se identifică noi funcţii se trece la descompunerea pe următorul nivel;
- procesul descompunerii continuă până toate modulele sunt terminale (un modul este terminal când nu se mai poate descompune în alte module).

Descompunerea are la bază următoarele reguli:

- nivelul 0 sau punctul iniţial de pornire îl constituie un modul neterminal sau coordonator;
- pentru toate modulele neterminale ale sistemului se aplică descompuneri succesive, în paşi, de sus în jos;
- descompunerea este terminată când modulele ultimului nivel sunt terminale.

Aplicarea în practică prezintă dezavantaje generate de definirea modelului de ansamblu a sistemului informatic pe baza unei analize complexe cu personal numeros, ceea ce determină prelungirea termenului de dare în exploatare a sistemului, iar erorile în definirea structurii şi a relaţiilor dintre module pot afecta toată activitatea ulterioară.

Metoda ascendentă BOTTOM-UP constă în agregarea modulelor de jos în sus punând în evidenţă legăturile dintre ele până se ajunge la un singur modul.

Conceptele care stau la baza acestei metode sunt ca şi la metoda descendentă: modularizarea şi abordarea sistemică.

Regulile care stau la baza metodei sunt:

- nivelul de agregare iniţial este nivelul la care se află modulele terminale;
- agregarea se face succesiv de jos în sus;
- când se obţine un nivel de agregare se realizează integrarea modulelor de nivel inferior în module de nivel superior;
- agregarea este terminată când la un nivel de agregare se obţine un singur modul.

Aplicarea acestei metode prezintă avantajul că sistemul se dezvoltă treptat, în concordanţă cu cerinţele reale ale utilizatorilor. Agentul economic poate beneficia mai repede de rezultatele prelucrării automate a datelor, se familiarizează cu sistemul în mod gradat, se reduc riscurile realizării unor sisteme de mare anvergură, neoperaţionale.

Dezavantajele acestei metode rezultă din gradul de integrare redus a modulelor datorită lipsei unei concepţii iniţiale de ansamblu, ceea ce face necesară reproiectarea unor componente.

4.3. ANALIZA SISTEMULUI INFORMAŢIONAL EXISTENT

4.3.1. Conceptul de analiză a sistemelor informaţionale

În general, analiza se defineşte ca fiind descompunerea reală sau mintală a unui obiect, fenomen sau a relaţiilor dintre obiecte, fenomene etc, în părţi componente în scopul cunoaşterii. Ea presupune deci examinarea amănunţită, parte cu parte, a unei probleme, a unui obiect de studiu.

Analiza sistemelor informaţionale economice este activitatea prin care se realizează cunoaşterea sistemului obiect si a cerinţelor de informaţii din sistemul de decizie. Vom numi sistem obiect orice parte a realităţii (a sistemului informaţional economic, în cazul nostru) care generează date şi care posedă entităţi, ce poate prelucra şi atribui semnificaţie datelor, parte delimitată în vederea realizării unui nou sistem informaţional.

Scopul analizei sistemului informaţional ar putea fi unul din următoarele:

1) proiectarea unui model al sistemului existent, adică a unui sistem sinonim cu ajutorul căruia să se simuleze funcţionarea sistemului real;
2) proiectarea unui sistem care să înlocuiască sistemul existent, având însă performanţe superioare;
3) proiectarea unui sistem raţionalizat:
    a) care pe baza aceloraşi intrări elaborează aceleaşi ieşiri, dar cu o structură mai simplă; de exemplu se raţionalizează structura organizatorică în paralel cu raţionalizarea sistemului de evidenţă;
    b) care simplifică intrările, structura şi in cazul în care se dovedeşte necesar, ieşirile sistemului.

Exista două concepte care stau la baza stabilirii elementelor sistemului obiect, care aplicate la activitatea de analiză identifică două tipuri de structuri:

- conceptul orientat pe organism, care determina structura organizatorică; în acest caz analiza sistemului informaţional se va face pe compartimente funcţionale;
- conceptul orientat pe activitate, care determină structura funcţională; caz în care analiza sistemului informaţional se va face pe funcţii si subfuncţii ale agentului economic analizat.

În general sunt studiate ambele tipuri de structuri, accentul punându-se însă pe activităţi, in timp ce elementele organizatorice interesează doar prin prisma aportului lor la realizarea activităţilor.

Activitatea de analiza a unui sistem informaţional economic se poate descompune in următoarele operaţii:

- culegerea datelor
- sistematizarea datelor culese;
- evaluarea sistemului informaţional existent.

În vederea delimitării corecte a cerinţelor informaţionale se pot distinge două variante de abordare a sistemului obiect, si anume de la analiza deciziilor de la analiza datelor:

Abordarea pornind de la analiza deciziilor caută să determine cerinţele de informaţii pornind de la analiza obiectivelor şi a deciziilor ce trebuie luate. Sunt culese şi investigate doar acele informaţii care sunt necesare modelului de decizie, şi anume:

- identificarea obiectivelor şi a deciziilor curente si potenţiale, precum si a atributelor conducerii, corespunzătoare obiectivelor si deciziilor;
- identificarea sau construirea unui model, a unei proceduri de elaborare a fiecărei decizii sau a unui model al procesului de luare a deciziilor;
- testarea senzitivităţii modelului la acurateţea si disponibilitatea datelor de intrare, specificarea limitelor si disponibilităţilor datelor cerute de model.

Avantajul acestui mod de abordare este acela că rezulta un volum mic de date de prelucrat.

În **abordarea pornind de la analiza datelor** se caută să se obţină cerinţele informaţionale prin analiza datelor folosite curent in sistemul existent, sau a datelor potenţial folosibile.

Sunt identificate datele colectate din sistemul existent, pentru care s-a perceput o necesitate, cât şi datele necolectate în sistemul existent, dar semnalate ca fiind utile, şi anume:

- cercetarea tuturor documentelor, fişierelor folosite in mod curent şi identificarea datelor care sunt culese si prelucrate in mod curent;
- identificarea datelor suplimentare, care nu sunt culese şi prelucrate în mod curent, dar care se dovedesc a fi utile;
- stabilirea datelor inutile, care se culeg în mod curent si nu sunt utilizate.

Acest mod de abordare pleacă de la premisa că toate datele utile trebuie să facă parte din sistem, pentru a preveni schimbările ce pot surveni în mediul agentului economic studiat, în stilul de decizie sau în cerinţele informaţionale.

Abordarea pornind de la analiza datelor conduce la un volum mai mare de date, dar are avantajul că deciziile ce se vor lua pe baza acestor date vor fi mai puţin afectate de modificările ce pot surveni pe parcurs.

Fiecare dintre cele două abordări este mai potrivită pentru un anumit tip de probleme, aplicaţii sau sisteme, analiza deciziilor se recomandă în special la activităţile de conducere, în cazul sistemelor informaţionale de management şi în cazul sistemelor suport de decizie, iar analiza datelor se recomandă în special în cazul activităţilor de rutină, în cazul sistemelor de gestiune.

4.3.2. Obiectivele şi fazele analizei sistemului informaţional existent (ASIE)

Această etapă de realizare a unui sistem informatic mai este cunoscută şi sub denumirea de *elaborarea temei de realizare*, *analiza diagnostic* sau *studiul de oportunitate*.

Această etapă are următoarele obiective :

1) Delimitarea ariei de cuprindere a sistemului informaţional existent care va deveni sistem obiect pentru conceperea şi realizarea unui sistem informatic. (Vom numi sistem obiect acea parte a realităţi economice care generează date şi care posedă entităţi ce pot prelucra şi atribui semnificaţie datelor, parte delimitată în vederea realizării unui sistem informatic).

2) Reflectarea activităţilor şi operaţiilor de culegere, transmitere şi prelucrare a datelor specifice sistemului informaţional existent.

3) Surprinderea modificărilor ce se impun în organizarea şi funcţionarea sistemului informaţional existent în viziunea conceperii unui sistem informatic.

4) Evidenţierea dotării cu tehnică de calcul, prin prisma concordanţei dintre sistemele electronice de calcul şi viitorul sistem informatic, inclusiv identificarea problemelor ce urmează a fi rezolvate de noul sistem cu ajutorul calculatorului electronic.

5) Fundamentarea unei soluţii de principiu care să precizeze activităţile şi operaţiile ce vor fi informatizate, costul antecalculat al sistemului, tipul calculatorului electronic, sistemul de operare aferent, inclusiv soluţia de gestiune a datelor (sistem de gestiune a fişierelor sau sistem

de gestiune a bazelor de date) pe care se va baza conceperea şi realizarea noului sistem, implicaţiile în organizarea internă a unităţii, precum şi planificarea globală a realizării sistemului informatic.

ASIE este necesară pentru a fundamenta direcţiile de perfecţionare a sistemului informaţional existent şi înlocuirea acestuia cu un sistem informatic care să satisfacă toate cerinţele informaţionale ale agentului economic. De asemenea analiza de sistem este oportună şi în cazul agenţilor economici nou înfiinţaţi pentru fundamentarea proiectului de sistem informaţional, ca parte a proiectului de înfiinţare a unităţii.

ASIE trebuie să ofere conducerii agentului economic informaţiile necesare fundamentării deciziei de perfecţionare a sistemului informaţional şi de proiectare a unui sistem informatic.

ASIE se desfăşoară în următoarele faze :

1) ***Organizarea şi conducerea analizei sistemului informaţional prin:*** pregătirea condiţiilor necesare analizei; constituire colectivului pentru ASIE; elaborarea programului de realizare;
2) ***Realizarea ASIE prin:*** documentarea pentru analiza de sistem; alegerea tehnicilor de analiză a sistemului informaţional existent; studiul componentelor sistemului informaţional existent; evaluarea critică a sistemului informaţional existent; elaborarea variantelor de realizare a sistemului informatic.
3) ***Finalizarea ASIE prin:*** definitivarea documentaţiei ASIE; avizarea ASIE de către beneficiar.

4.3.3. Organizarea şi conducerea ASIE

Pregătirea condiţiilor necesare ASIE. Iniţierea şi declanşarea analizei de sistem este atributul conducerii unităţii beneficiare care precizează tema şi obiectivele analizei printr-o Notă de comandă. Pe baza Notei de comandă se încheie Contractul de execuţie pentru ASIE între unitatea beneficiară şi unitatea de analiză proiectare. Valoarea contractului se determină pe baza unui deviz în care se stipulează: numărul de ore/om preconizat pentru executarea analizei, personalul implicat, tarifele de realizare, termenele de execuţie, inclusiv obligaţiile părţilor şi modul de decontare a lucrărilor.

Constituirea colectivului pentru ASIE. Colectivul poate fi alcătuit din :

- personalul propriu al unităţii beneficiare;
- personalul unei unităţi specializate în analiza şi proiectarea de sisteme informatice;
- din personalul unităţii beneficiare şi al unei unităţi specializate.

Prima variantă prezintă avantajul cunoaşterii sistemului informaţional de către colectivul de analiză, ceea ce asigură efectuarea unei analize de calitate într-un termen relativ scurt, dar prezintă dezavantajul că membrii colectivului nu pot sesiza toate deficienţele şi limitele reale ale sistemului informaţional, deoarece sunt familiarizaţi cu structura şi funcţionarea acestuia.

Varianta a doua prezintă avantajul că membrii colectivului de analiză sunt specializaţi în rezolvarea unor astfel de probleme, cunosc deficienţele şi limitele unor sisteme similare cu cel analizat, dar prezintă dezavantajul realizării analizei într-un termen relativ îndelungat, deoarece se impune o cunoaştere detaliată a unor anumite părţi cu caracter specific, de multe ori unicat, ale sistemului informaţional.

Varianta a treia este cea mai indicată datorită maximizării avantajelor şi minimizării dezavantajelor primelor două variante.

În orice variantă, pe lângă analiştii de sistem, din componenţa colectivului trebuie să facă parte conducători şi membri ai compartimentelor funcţionale supuse analizei. Aceasta deoarece conducătorii compartimentelor funcţionale pot identifica informaţiile utile, pe cele de prisos şi pot formula cerinţe informaţionale pentru noul sistem, iar personalul de specialitate din compartimentele funcţionale, cunoscând detaliat toate activităţile şi operaţiile desfăşurate, poate sesiza neajunsurile segmentului de sistemului informaţional pe care lucrează.

Numărul membrilor colectivului depinde de obiectivele analizei, de sfera de cuprindere, de complexitatea şi particularităţile sistemului informaţional şi de termenele de realizare.

Elaborarea programului de realizare. Desfăşurarea activităţii de analiză a sistemului se efectuează pe baza unui program prin care se urmăreşte folosirea raţională a personalului, termenele intermediare şi finale, precum şi procedurile de control şi avizare a realizării ASIE. Programarea realizării ASIE se poate face prin folosirea grafului de tip PERT iar urmărirea operativă a execuţiei se poate efectua prin intermediul Graficelor de tip GANTT.

4.3.4. Realizarea analizei sistemului informaţional existent

Documentarea pentru ASIE. Activitatea de documentare necesară analizei de sistem presupune studiul unor probleme de organizare ale agentului economi, a particularităţilor procesului tehnologic, a fluxurilor informaţionale şi a activităţilor desfăşurate, precum şi a cadrului legislativ impus funcţionării sistemului informaţional.

În această viziune se studiază modul de organizare şi conducere a agentului economic; obiectul activităţii de bază; particularităţile procesului tehnologic; strategia şi tactica conducerii în realizarea activităţilor de bază; principalele acţiuni pentru dezvoltarea unităţii; etc.

În continuare se studiază actele normative care reglementează activitatea agentului economic în ansamblu şi activitatea fiecărui compartiment funcţional precum şi modul în care se respectă cadrul legal şi deciziile de funcţionare efectivă a sistemului informaţional al unităţii analizate.

Documentarea este bine să fie completată cu studiul literaturii de specialitate şi a celor mai noi realizări în organizarea şi funcţionarea sistemelor informaţionale, precum şi cu experienţa pozitivă din unităţi care fac parte din aceeaşi ramură de activitate cu unitatea studiată.

Alegerea tehnicilor de analiză. Analiza sistemelor informaţionale se poate realiza utilizând mai multe tehnici. Astfel distingem doua categorii de tehnici de analiză a sistemelor informaţionale economice :

- tehnici elementare;
- tehnici complexe.

Tehnicile elementare realizează doar operaţia de culegere a datelor şi asigură o sistematizare redusă a acestora. În grupa tehnicilor elementare de analiza se încadrează:

- observarea directa;
- studierea documentaţiei existente;
- participarea analistului la efectuarea lucrărilor din sistemul informaţional;
- interviul;
- chestionarul.

Tehnicile complexe realizează toate activităţile de analiza, respectiv culegerea şi sistematizarea datelor, precum şi evaluarea sistemului informaţional. Culegerea datelor, în cadrul tehnicilor complexe de analiză, se realizează utilizând, de regulă, mai multe tehnici elementare.

Sistematizarea datelor, constând nu doar intr-o simplă ordonare a lor, ci şi într-o sintetizare, se realizează folosind una sau mai multe tehnici de reprezentare. Dintre tehnicile de reprezentare amintim:

- scrierea tehnica;
- reprezentarea grafică a fluxurilor informaţionale prin: organigrame (scheme logice), schema bloc a fluxurilor informaţionale (diagrama de rutina sau diagrama de relaţii), schema orizontală a fluxurilor informaţionale, schema verticală a fluxurilor informaţionale;
- grilele;
- listele de acţiuni condiţionate;
- tabelele de decizii;
- limbajul pseudocod etc.

Dintre tehnicile complexe de analiza a sistemelor informaţionale fac parte:

- analiza diagnostic;
- analiza celulara;
- analiza S.O.P. (Study Organization Plan).

În general, alegerea uneia sau alteia din tehnicile elementare de investigare a sistemului informaţional existent şi din tehnicile de reprezentare se face în funcţie de: complexitatea şi particularităţile sistemului informaţional existent; aria de cuprindere a acestuia; condiţiile concrete de lucru; termenele de realizare; experienţa analiştilor.

Practica a cristalizat câteva recomandări privind alegerea şi utilizarea tehnicilor de analiză:

1) Nici o tehnică nu poate, ea singură, să asigure culegerea tuturor datelor pentru caracterizarea complexă a sistemului informaţional existent, de aceea se impune utilizarea mai multor tehnici. De regulă se utilizează ca tehnică principală interviul completat cu chestionarul sau cu inventarul documentelor.

2) În vederea alegerii corecte a tehnicii de culegere a datelor şi stabilirii modului de realizare, se impune o bună documentare prealabilă privind unitatea ce va fi supusă analizei, în special personalul său.

3) În culegerea datelor analistul nu va avea idei preconcepute despre problemele unităţii şi despre modul de rezolvare a lor, deoarece, în loc să descopere adevăratul mecanism al activităţii investigate, va primi doar confirmarea propriilor păreri şi construirea proiectului pe această bază va conduce sigur la eşec.

4) În timpul culegerii datelor nu se elaborează soluţii, eventualele idei de soluţii se notează fără a insista asupra lor (din acelaşi motiv ca în cazul anterior).

Studiul componentelor sistemului informaţional existent. Analiza elementelor componente ale sistemului informaţional următoarele obiective:

a) identificarea caracteristicilor generale ale agentului economic;

b) studiul activităţilor desfăşurate de agentul economic;

c) studiul deciziilor din sistemul agentului economic;

d) studiul dotării cu tehnică de calcul;

e) studiul fluxurilor informaţionale;

f) studiul volumului datelor;

g) studiul costului de funcţionare a sistemului informaţional.

a) În cadrul primului obiectiv se studiază: profilul şi obiectivele agentului economic, cadrul juridic de funcţionare, dinamica unor indicatori tehnico-economici privind capacitatea de producţie, gradul de folosire a utilajelor, cifra de afaceri, numărul de salariaţi, metodele de previziune, forma de contabilitate, metoda de evidenţă a valorilor materiale, metoda de calculaţie a costurilor, relaţiile informaţionale ale unităţii analizate cu alţi agenţi economici şi cu organe ierarhice superioare şi inferioare, determinarea modului de ierarhizare a secţiilor de producţie şi a compartimentelor funcţionale prin organigrama structurii organizatorice.

b) În cadrul celui de-al doilea obiectiv se studiază: funcţiile unităţii pe baza statutului de funcţionare: activităţile şi sarcinile din cadrul funcţiilor, atribuţiile fiecărui post de lucru, dar şi funcţia de bază a agentului economic.

c) Studiul deciziilor din sistemul agentului economic urmăreşte modul cum sunt fundamentate aceste decizii documentele existente şi informaţiile de bază.

d) Studiul dotării cu tehnică de calcul permite analiza dotării cu tehnică de calcul, gradul ei de utilizare şi măsura în care aceasta contribuie la funcţionarea sistemului informaţional existent.

Se analizează modul în care configuraţia existentă răspunde cerinţelor actuale din punct de vedere al capacităţii de stocare şi prelucrare, precum şi posibilităţile de integrare în noul sistem.

e) Studiul fluxurilor informaţionale verifică circuitul documentelor având în vedere:

- descrierea fluxurilor informaţionale printr-o organigramă unde se precizează: activităţile, compartimentele sau posturile de lucru, procedurile efectuate asupra documentelor şi circuitul documentelor între posturile de lucru;

- evidenţierea fluxurilor paralele de date şi a circuitelor neraţionale care trebuie eliminate din noul sistem;
- identificarea documentelor vehiculate în mod inutil sau care circulă într-un număr de exemplare necorespunzător cerinţelor reale ale sistemului informaţional;
- determinarea gradului de utilizare a documentelor tipizate pentru eliminarea din sistem a documentelor netipizate;
- determinarea gradului de încărcare şi solicitare a fiecărui compartiment şi post de lucru implicat în sistemul informaţional;
- integrarea sistemului informaţional analizat cu alte sisteme informaţionale.

f) Studiul volumului datelor se referă la evidenţierea şi cuantificarea următoarelor elemente: simbolul şi denumirea documentelor, frecvenţa de întocmire a documentelor, numărul maxim şi mediu de documente corespunzător frecvenţei de întocmire a documentelor, numărul maxim şi mediu de rânduri pe document, numărul maxim şi mediu de caractere pe rând, numărul de exemplare pe document, estimarea numărului mediu de documente pentru perioada viitoare.

Concomitent cu analiza documentelor se urmăreşte sistemul de coduri pentru a fi preluat în noul sistem informatic deoarece activitatea de codificare este complexă şi personalul din unitate este familiarizat cu codurile existente.

Pe baza evaluărilor făcute, echipa de analiză poate formula recomandări privind tipul şi capacitatea perifericelor, suporturile tehnice de date necesare în sistem şi poate face o evaluare critică a sistemului informaţional existent.

g) Studiul costului de funcţionare al sistemului informaţional se referă la analiza cheltuielilor cu salariile personalului implicat în funcţionarea sistemului, a cheltuielilor cu amortizarea şi întreţinerea tehnicii de calcul, a cheltuielilor cu materiale consumabile. Identificarea costului de funcţionare a sistemului informaţional existent serveşte în faza de analiză de sistem pentru a aprecia performanţele acestuia şi după implementarea sistemului pentru calculul eficienţei economice a noului sistem când se compară cheltuielile cu efectele noului şi vechiului sistem.

Evaluarea sistemului informaţional. Prin aceasta se urmăreşte evaluarea performanţelor şi limitelor sistemului informaţional în raport cu cerinţele sistemului de conducere şi evaluarea gradului de pregătire a unităţii analizate pentru realizarea unui sistem informatic.

Evaluarea performanţelor unui sistem informaţional se face pe baza următoarelor criterii:

- măsura în care sistemul informaţional asigură realizarea obiectivelor şi îndeplinirea funcţiilor de bază ale unităţii;
- gradul de asigurare cu informaţiile necesare la diferite niveluri de conducere;
- operativitatea în culegerea, prelucrarea datelor, transmiterea informaţiilor şi adoptarea deciziilor prin prisma timpului de răspuns;
- calitatea informaţiilor rezultate din prelucrare, privită sub aspectul preciziei de exprimare, al flexibilităţii şi frecvenţei de difuzare;
- calitatea şi siguranţa legăturilor între posturile de lucru în cadrul fluxurilor informaţionale;
- posibilităţile sistemului informaţional de a sesiza tendinţele în evoluţia unităţii;
- posibilităţile de control şi efectuare de corecţii, reacţia la apariţia unor evenimente externe, posibilităţile de corectare în timp util a abaterilor;
- gradul de integrare a sistemului informaţional sub aspectul reducerii informaţiilor redundante, a compatibilităţii informaţiilor cu datele de intrare, a organizării datelor în baze de date unitare şi posibilitatea de asigurare a unui cadru organizatoric adecvat;
- gradul de automatizare a lucrărilor în totalul lucrărilor din sistemul informaţional.

Evaluarea gradului de pregătire a unităţii analizate pentru proiectarea unui sistem informatic are în vedere: pregătirea personalului, disciplina tehnologică în sistemul

informaţional, existenţa cadrului organizatoric, existenţa bazei ştiinţifico-metodologice şi informaţionale.

Concret, evaluarea sistemului informaţional trebuie să facă referiri la:

- obiectivele sistemului;
- mijloacele şi metodele de prelucrare;
- personalul implicat;
- costurile de funcţionare ale sistemului.

Evaluarea critică a obiectivelor urmăreşte să determine modul în care sunt rezolvate cerinţele informaţionale din sistem, măsura în care informaţiile solicitate lipsesc sau sunt necorespunzătoare. Trebuie pusă în evidenţă concordanţa dintre structura organizatorică şi categoriile de informaţii necesare fiecărui post. Existenţa în sistemul informaţional a unor informaţii de prisos este la fel de dăunătoare ca şi lipsa de informaţii, deci, trebuie făcută o inventariere exactă a informaţiilor din sistem, evidenţiate procedurile asemănătoare de prelucrare a datelor şi fluxurile paralele de date.

Evaluarea critică a mijloacelor tehnice utilizate trebuie să pună în evidenţă gradul de dotare cu tehnică de calcul, ponderea acesteia în sistemul informaţional, gradul ei de utilizare şi oportunitatea folosirii tehnicii existente în noul sistem. Se poate analiza gradul de flexibilitate al tehnicii de calcul şi măsura în care aceasta este compatibilă cu exigenţele actuale şi previzibile ale sistemului. Se pot face evaluări privind gradul de uzură fizică sau morală a sistemului de calcul.

Evaluarea critică a personalului are rolul de a determina gradul de utilizare al timpului de lucru şi gradul de încărcare pe fiecare compartiment şi post de lucru. Se poate analiza dacă structura personalului pe profesii şi niveluri de pregătire este în concordanţă cu exigenţele impuse de specificul sistemului informaţional. Se pot face aprecieri asupra gradului de perfecţionare profesională a personalului.

Concret, se poate determina numărul total de personal implicat în funcţionarea sistemului informaţional şi fondul de timp maxim disponibil alocat, exprimat în ore-muncă, numărul total de documente sau informaţii vehiculate în sistem raportate la numărul total de personal, ponderea fondului de timp maxim disponibil alocat personalului ce lucrează în sistemul informaţional în totalul fondului de timp de la nivelul agentului economic, productivitatea muncii pe persoană, timpul mediu pentru completarea unui document, ponderea salariaţilor care lucrează în sistemul informaţional în totalul salariaţilor şi dinamica acestor indicatori.

Evaluarea costului de funcţionare a sistemului informaţional presupune punerea în relief a măsurii pentru reducerea cheltuielilor cu funcţionarea sistemului informaţional. Este indicată precizarea efectelor negative care apar datorită lipsei din sistem a unor informaţii şi estimarea lor valorică. Se pot calcula următorii indicatori cu privire la costul sistemului informaţional: cheltuieli cu informaţiile strict necesare; cheltuieli cu informaţiile suplimentare; cheltuieli necesare pentru utilizarea a "n" informaţii; cheltuieli cu informaţiile la 1000 de lei volum de activitate; costul mediu de utilizare pe un document .

Elaborarea variantelor de realizare a noului sistem. Presupune parcurgerea următoarelor activităţi:

I) stabilirea funcţiilor, cerinţelor şi restricţiilor noului sistem;

II) definirea soluţiei globale de principiu pentru prelucrarea automată a datelor;

II) evaluarea fiecărei variante de realizare.

I) Stabilirea funcţiilor, cerinţelor şi restricţiilor noului sistem se bazează pe evaluarea critică a sistemului existent şi pe cerinţele beneficiarului.

Stabilirea funcţiilor noului sistem presupune delimitarea ariei de cuprindere a principalelor activităţi pe care le va acoperi noul sistem, legătura cu aplicaţiile informatice în exploatare sau prevăzute a se realiza în viitor.

Subsistemele, aplicaţiile informatice care se pot fi proiectate şi realizate într-un sistem economic sunt:

1. Subsistemul pentru activitatea de planificare tehnico-economică:

- elaborarea planului anual: stabilirea variantelor de plan, calculul indicatorilor tehnico-economici, alegerea variantei optime. Se elaborează studii de prognoză privind cerinţele pieţei, se întocmesc mai multe variante urmărind valorificarea potenţialului productiv existent şi crearea resurselor necesare pentru realizarea lor.
- defalcarea planului pe trimestre, luni, subunităţi care se face în funcţie de nivelul resurselor, termenele contractuale şi priorităţile beneficiarului. Se recalculează valorile indicatorilor economico-financiari.
- urmărirea şi analiza realizării planului. Se centralizează pe baza rapoartelor de producţie realizările şi se semnalează abaterile, se calculează indicatorii economici şi se semnalează abaterile.

2. **Subsistemul pentru pregătirea tehnică a producţiei**:

- elaborarea şi actualizarea nomenclatoarelor de produse pe baza informaţiilor din documentaţia constructivă a produselor;
- elaborarea şi actualizarea fişierelor tehnologice pe baza precizărilor din documentaţia tehnologică pe fiecare produs;
- calculul consumului specific de manoperă pe baza documentaţiei specifice, pe meserii, utilaje, diferite niveluri de agregare a produselor;
- calculul loturilor optime şi ciclurilor de fabricaţie pe baza documentaţiei tehnice;
- calculul consumurilor specifice de materii prime, materiale, SDV;
- urmărirea şi analiza încadrării în normele de consum;
- urmărirea şi analiza planului privind realizarea produselor noi.

3. Subsistemul de programare, lansare şi urmărire a producţiei de bază:

- calculul necesarului de fabricat pe o perioadă pe baza planului de producţie şi a nomenclatoarelor de produse;
- calculul fondului de timp al utilajelor pe baza indicatorilor privind starea utilajelor şi planul de reparaţii;
- elaborarea programelor de fabricaţie trimestriale şi lunare în funcţie de termenele de livrare, disponibilul de capacităţi, forţa de muncă şi resursele materiale;
- programarea operativă a producţiei, detalierea sarcinilor din programul de fabricaţie la nivel de decadă, zi, schimb, pe ateliere, grupuri de maşini pentru stabilirea unor loturi optime de fabricaţie;
- lansarea manoperei care presupune calculul şi centralizarea consumului de timp necesar executării operaţiilor pentru producţie şi emiterea bonurilor de manoperă;
- lansarea materialelor, calculul necesarului de materiale, emiterea bonurilor de consum sau a fişelor limită de consum;
- controlul calităţii producţiei, urmărirea şi analiza realizării producţiei.

4. Subsistemul pentru aprovizionarea tehnico-materială:

- calculul normativelor de materii prime şi materiale;
- planul de aprovizionare pe baza planului de producţie şi a normelor de consum;
- calculul necesarului pentru aprovizionat cu defalcarea pe trimestre şi luni pe baza planului de aprovizionare şi a stocului existent la începutul perioadei;
- lansarea şi urmărirea comenzilor de aprovizionare;
- urmărirea gradului de acoperire a necesarului de aprovizionat;
- analiza şi urmărirea realizării activităţii de aprovizionare.

5. Subsistemul pentru activitatea de desfacere:

- elaborarea de studii de previziune privind cerinţele pieţei;
- evidenţa şi analiza comenzilor primite de la beneficiari;
- urmărirea gradului de acoperire a planului de producţie cu comenzi şi contracte;
- întocmirea planului anual de livrare;
- urmărirea derulării contractelor cu beneficiarii;
- urmărirea şi analiza privind activitatea de desfacere.

6. Subsistemul pentru gestiunea valorilor materiale:
- gestiunea stocului de materii prime şi materiale;
- gestiunea stocului de produse finite;
- gestiunea stocului de semifabricate proprii;
- urmărirea şi analiza stocurilor supranormative şi a celor cu mişcare lentă.

7. Subsistemul pentru activitatea de personal:
- elaborarea planului forţei de muncă, calculul fondului de salarii pe baza planului de producţie, a normelor tehnologice;
- evidenţa şi structura personalului: elaborarea situaţiei privind existentul şi prezenţa la lucru a personalului, fluctuaţia personalului;
- calculul salariilor;
- analiza şi raportarea forţei de muncă.

8. Subsistemul pentru activitatea financiar-contabilă:
- elaborarea bugetului de venituri şi cheltuieli;
- evidenţa mijloacelor fixe: urmărirea pe locuri de folosinţă, calculul şi includerea pe costuri a amortizării, urmărirea uzurii fizice şi morale;
- contabilitatea valorilor materiale;
- contabilitatea salariilor: întocmirea formelor de plată: lista de avans chenzinal, ştatul de salarii, indemnizaţii pentru concediul de odihnă, plăţile în contul asigurărilor sociale;
- calculaţia costurilor: înregistrarea corectă şi la timp pe locuri de folosinţă şi obiecte de calculaţie a cheltuielilor de producţie, calculul costurilor unitare pe produs, analiza costurilor de producţie.
- contabilitatea generală cu: evidenţa mijloacelor băneşti şi a împrumuturilor, debitori, creditori, decontările cu furnizorii, cu beneficiarii, cu bugetul de stat, cu asigurările sociale, veniturile şi rezultatele financiare; întocmirea balanţei de verificare şi a situaţiilor de raportare; întocmirea bilanţurilor şi anexelor sale.

9. Subsistemul pentru activităţi auxiliare sau specifice:
- activitatea de întreţinere şi reparaţii utilaje;
- activitatea de realizare a pieselor de schimb;
- activitatea de transport.

Restricţiile viitorului sistem reprezintă condiţii impuse proiectantului de către beneficiar sau cadru legal. Ele pot fi:
- de natură legislativă: proiectantul trebuie să cunoască actele normative ce reglementează activitatea în sistemul obiect şi să ţină seama de prevederile acestora în realizarea noului sistem;
- de natură organizatorică: nu există specialişti în informatică şi până la formarea lor se apelează la o unitate specializată;
- de natură tehnică: viitorul sistem nu se poate realiza decât cu o configuraţie de calcul şi cu un soft de bază;
- cu caracter general: proiectul trebuie terminat la o anumită dată şi cu un anumit cost.

II) Definirea unor soluţii globale de principiu pentru prelucrarea automată a datelor are în vedere obiectivele stabilite anterior prin definirea mai multor soluţii de prelucrare automată în noul sistem. Pentru fiecare variantă se stabilesc numărul activităţilor şi gradul de informatizare a acestora, delimitându-se operaţiile ce se efectuează automat de cele care se vor efectua manual. Se vor preciza atribuţiile preluate de sistemul informatic în raport cu cele ce vor rămâne pe seama compartimentelor funcţionale. Avându-se în vedere aceste opţiuni cu caracter funcţional se studiază soluţiile tehnice de principiu pentru:

a) Posibilităţile financiare ale agentului economic de a dispune de configuraţia de calcul necesară. Echipamentul poate fi propriu sau închiriat;

b) Felul calculatoarelor: mari, medii, mini, micro şi modul lor de folosire. Interconectarea într-o reţea, chiar eterogenă, cu regim de exploatare distribuit local sau la distanţă, cu staţii de lucru bazate pe calculatoare personale are avantajul creşterii numărului de utilizatori, exploatării intensive a capacităţii de prelucrare şi satisfacerii în timp real a cerinţelor utilizatorilor, dar este mai costisitoare;

c) Tipul sistemului de operare şi softului de bază minim pentru funcţionarea sistemului;

d) Felul produselor-program ce vor fi folosite: pot fi produse-program specifice sau generalizate. Prima soluţie are avantajul posibilităţii de informatizare a problemelor complexe specifice fiecărui beneficiar, dar are dezavantajul că necesită un personal strict specializat. A doua variantă scurtează durata de realizare şi costul este relativ scăzut, dar are dezavantajul limitării informatizării la facilităţile oferite de produsul-program şi este puţin flexibilă;

e) Tipul suporturilor tehnice de date folosite în noul sistem şi modalitatea concretă de introducere a datelor în sistemul de calcul şi obţinerea rezultatelor. Soluţia cea mai eficientă de introducere a datelor este preluarea directă de la terminal. Afişarea rezultatelor se poate face la imprimantă pentru situaţiile de ieşire ce vor servi ca documente justificative, dar cu costuri mari şi un timp mare de obţinere a rezultatelor, sau se poate face la videoterminal;

f) Modul de organizare a datelor:

- în fişiere independente: se realizează uşor, redundanţă mare a datelor, limitarea ieşirilor la datele stocate în fişiere, dependenţa fizică şi logică a programelor de aplicaţie faţă de modul de descriere a datelor;
- în baze de date gestionate de un sistem de gestiune a bazelor de date;
- în baze de date distribuite local sau la distanţă care oferă facilităţi superioare în actualizarea şi exploatarea datelor, dar cu eforturi mai mari;
- în baze de cunoştinţe care oferă avantaj din punct de vedere a posibilităţilor de prelucrare şi dezavantaj din punct de vedere al modului de realizare şi costurilor mari.

g) Tipul de prelucrare a datelor: pe loturi, interactivă, centralizată fără teleprelucrare şi centralizată cu teleprelucrare distribuită. Prelucrarea pe loturi este avantajoasă pentru prelucrări multiple efectuate o singură dată la sfârşitul unei perioade de gestiune, dar prezintă dezavantajul lipsei de operativitate a informaţiilor. Prelucrarea interactivă are avantajul realizării unor sisteme informatice dinamice, asigură obţinerea de informaţii în timp real, dar în anumite sisteme de calcul nu folosesc intensiv capacitatea de prelucrare. Prelucrarea centralizată fără teleprelucrare asigură exercitarea unui bun control al prelucrării, o bună securitate privind accesul la date, dar influenţează negativ timpul de răspuns al sistemului. Prelucrarea centralizată cu teleprelucrare este cea mai avantajoasă, dar necesită costuri mari de realizare.

III) Evaluarea fiecărei variante de realizare completează definirea funcţionalităţii fiecărei variante cu o serie de informaţii referitoare la cheltuielile ocazionate şi efectele economice scontate. Această comparare va permite beneficiarului să selecteze varianta cea mai eficientă. Evaluarea costurilor va urmări, atât cheltuielile prevăzute pentru proiectarea şi realizarea sistemului informatic, cât şi pe cele necesare exploatării curente a acestuia.

La prezentarea efectelor economice se vor avea în vedere, atât efectele economice cuantificabile, cât şi efectele economice necuantificabile.

Evaluarea termenelor de realizare are în vedere precizarea intervalelor şi termenelor de proiectare a noului sistem, precum şi a momentelor de achiziţionare a sistemelor de calcul necesare. De asemenea, această evaluare trebuie să precizeze termenele şi intervalele de pregătire profesională a personalului din posturile de lucru ce vor fi implicate în noul sistem, cât şi a personalului din unitatea proprie de informatică.

Conducerea unităţii beneficiare evaluează şi selectează varianta cea mai convenabilă prin luarea în considerare a tuturor variantelor, prin consultarea unor specialişti în analiza şi realizarea de sisteme informatice sau (dacă este posibil) prin analiza unor soluţii de informatizare alese de unităţi similare.

4.4. PROIECTAREA GENERALĂ A SISTEMELOR INFORMATICE

4.4.1. Obiective şi activităţi specifice proiectării generale

Proiectarea generală are ca obiectiv elaborarea concepţiei logice a sistemului informatic, definirea acestuia din punct de vedere structural şi funcţional. Aceasta presupune stabilirea componentelor sistemului informatic, a ieşirilor, a bazei informaţionale de intrare, a documentelor pe care sunt consemnate datele de intrare, a legăturilor dintre ele şi a funcţionalităţii sistemului astfel încât toate elementele sale să formeze un întreg, să se articuleze ca un tot unitar.

Structura generală a sistemului informatic cuprinde un ansamblu de intrări, prelucrări şi ieşiri definite în funcţie de obiectivele noului sistem.

I. Intrările sistemului informatic cuprind :

- baza de date
- mişcările (tranzacţiile).

Baza de date ca noţiune generală reprezintă colecţii de date operaţionale, memorate şi utilizate în comun de către componentele sistemului informatic.

Tranzacţiile sunt reprezentate de datele care reflectă modificări şi devin componente ale bazei de date în urma clasificării, validării, înregistrării şi prelucrării lor.

În cadrul unui sistem informatic există două categorii de tranzacţii:

- tranzacţii externe;
- tranzacţii interne.

Tranzacţiile externe redau fenomenele şi procesele economice, inclusiv modificările produse în activitatea agentului economic, ca de exemplu: aprovizionarea cu un material, livrarea unui produs, încasarea unei facturi, modificarea preţului unui produs etc.

Tranzacţiile interne apar în momentul obţinerii unui rezultat, fiind destinate actualizării bazelor de date. De exemplu: totalul valorii facturilor primite de la un furnizor care actualizează la o anumită perioadă soldul furnizorului respectiv; totalul vânzărilor dintr-un produs într-o zi care actualizează stocul produsului respectiv etc.

Tranzacţiile externe provin din exteriorul sistemului de calcul şi sunt furnizate prin proceduri de introducere a datelor din sistem, iar tranzacţiile interne sunt asigurate exclusiv în cadrul sistemului de calcul prin proceduri de actualizare şi exploatare a colecţiilor de date create anterior.

II. Prelucrările sistemului informatic sunt asigurate prin execuţia unui ansamblu de programe care se subdivid în: programe pentru actualizarea bazelor de date şi programe pentru obţinerea rezultatelor şi a situaţiilor de ieşire proiectate.

Deci sistemele informatice funcţionează, la prima vedere, în două cicluri:

- actualizarea bazelor de date când se validează şi se introduc tranzacţiile;
- obţinerea situaţiilor de ieşire solicitate.

Prelucrările sistemului informatic acţionează asupra colecţiilor de date care sunt constituite într-un nucleu informaţional ce conţine mulţimea atributelor necesare obţinerii informaţiilor solicitate. Nucleul de informaţii urmează a fi structurat în entităţi informaţionale al căror ansamblu delimitează baza informaţională a sistemului informatic.

Baza informaţională este formată din ansamblul entităţilor informaţionale şi atributelor componente ale acestora ce descriu dinamica fenomenelor şi proceselor economice la un agent economic pe o perioadă de timp.

Procedurile de creare, actualizare, exploatare, listare şi colecţiile de date organizate în fişiere sau baze de date formează nucleul sistemului informatic.

III. Ieşirile sistemului informatic reprezintă rezultatul prelucrării colecţiilor de date şi se vor concretiza în situaţii de ieşire solicitate de beneficiar sau în colecţii de date: fişiere sau baze de date actualizate ce vor fi transmise altor sisteme sau aplicaţii informatice.

Proiectarea generală se derulează în mai multe faze:

- stabilirea obiectivelor pe care le va acoperi viitorul sistem;
- definitivarea conţinutului informaţional al ieşirilor conform cu obiectivele stabilite;
- determinarea bazei informaţionale necesare obţinerii ieşirilor proiectate;
- formalizarea atributelor de intrare şi a elementelor structurale ale sistemului informatic prin codificare;
- proiectarea documentelor de intrare;
- proiectarea structurală şi funcţională a noului sistem;
- elaborarea documentaţiei proiectării generale.

Deoarece asigură definirea conceptuală a sistemului informatic, proiectarea generală este independentă faţă de soluţiile tehnice impuse de tipul calculatoarelor electronice, software-ului utilizat şi modul de gestiune a bazelor de date.

Proiectarea generală se realizează în mai multe variante având în vedere aria de cuprindere a noului sistem, complexitatea obiectivelor, gradul de flexibilitate ce se cere noului sistem. În abordarea proiectării generale se poate pleca de la ieşiri la intrări, de la intrări la ieşiri sau se poate realiza o variantă mixtă.

În **varianta ieşiri - intrări** se pleacă de la obiectivele sistemului informatic şi situaţiile de ieşire unde se concretizează acestea.

Analizând modul de obţinere a fiecărei informaţii se determină baza informaţională de intrare, apoi se realizează celelalte faze ale proiectării.

Ordinea de parcurgere a fazelor este:
- stabilirea obiectivelor;
- proiectarea ieşirilor;
- proiectarea bazei informaţionale de intrare;
- codificarea;
- proiectarea documentelor de intrare;
- proiectarea structurală şi funcţională;
- elaborarea documentaţiei.

Varianta intrări - ieşiri poate avea la ieşire numai atributele existente în baza informaţională de intrare sau indicatorii ce se pot calcula din atribute.

Această variantă se recomandă în cazul realizării unor sisteme informatice mari şi mijlocii caracterizate prin complexitatea obiectivelor şi a ieşirilor informaţionale solicitate.

După stabilirea obiectivelor se inventariază documentele de intrare care circulă în sistemul agentului economic pentru determinarea bazei informaţionale de intrare.

Ordinea de parcurgere a fazelor este;
- definirea obiectivelor;
- inventariere atributelor de intrare şi a legăturilor dintre acestea pe baza documentelor de intrare folosite;
- proiectarea bazei informaţionale de intrare;
- codificarea;
- proiectarea documentelor de intrare;
- proiectarea ieşirilor sistemului informatic;
- proiectarea structurală şi funcţională;
- elaborarea documentaţiei.

Avantajul acestei variante este flexibilitatea bazei informaţionale de intrare în condiţiile modificării cerinţelor informaţionale. Dezavantajul este dat de baza informaţională de intrare care este supradimensionată, ceea ce determină un timp mare de realizare, costuri ridicate şi creşterea complexităţii prelucrărilor din sistemul informatic.

Varianta mixtă porneşte prin stabilirea obiectivelor noului sistem şi a unei variante iniţiale a bazei informaţionale de intrare, apoi avându-se în vedere posibilităţile de prelucrare se determină baza informaţională de ieşire, iar în funcţie de solicitările beneficiarului se proiectează ieşirile necesare în prezent şi în perspectivă şi pe baza lor se proiectează varianta finală a bazei informaţionale de intrare.

Ordinea de parcurgere a fazelor este:

- definirea obiectivelor;
- proiectarea iniţială a bazei informaţionale de intrare;
- codificarea;
- proiectarea documentelor de intrare;
- proiectarea ieşirilor sistemului informatic;
- reproiectarea bazei informaţionale de intrare;
- proiectarea structurală şi funcţională;
- elaborarea documentaţiei.

4.4.2. Proiectarea ieşirilor sistemului informatic

Realizarea practică a obiectivelor oricărui sistem informatic se concretizează prin satisfacerea cerinţelor informaţionale ale sistemului de conducere din organismul economic pe care acesta este grefat, respectiv prin furnizarea ieşirilor.

Ieşirile sistemului informatic pot fi privite din trei puncte de vedere:

- structural;
- funcţional;
- tipologic.

Din punct de vedere structural ieşirile sistemului informatic reprezintă a treia componentă din triada ce caracterizează structura oricărui tip de sistem, respectiv INTRĂRI - PRELUCRĂRI - IEŞIRI.

Din punct de vedere funcţional ieşirile sistemului informatic concretizează obiectivele generale şi specifice ale sistemului.

Din punct de vedere tipologic ieşirile sistemului informatic pot fi realizate sub formă de:

- liste (rapoarte sau situaţii) de ieşire care cuprind indicatori ai proceselor, stărilor şi rezultatelor activităţii economice;
- grafice, care redau într-o formă sinoptică starea şi evoluţia indicatorilor economici;
- ieşiri către alte sisteme informatice sub forma unor colecţii organizate de date (fişiere sau baze de date); acestea pot fi transmise indirect (off-line) prin intermediul suporturilor tehnice de date (disc magnetic, bandă magnetică, etc.) sau direct (on-line) prin mijloace de teletransmisie.

Într-un sistem informatic economic listele ce grupează indicatori economici constituie ponderea situaţiilor de ieşire, dar se constată, în ultimul timp, o creştere a ponderi ieşirilor sub formă grafică, proces ce se va amplifica pe măsura implicării crescânde a sistemelor informatice în procesul decizional.

În proiectarea situaţiilor de ieşire se va ţine seama de următoarele aspecte:

a) destinaţia situaţiei şi momentul când se estimează că se poate obţine, nivelul de detaliere a informaţiei şi anume: informaţii mai analitice la nivelurile inferioare ale conducerii şi informaţii de sinteză la nivelul superior;

b) existenţa în raport a tuturor informaţiilor necesare beneficiarului pe segmentul stabilit;

c) aceeaşi informaţie să nu apară în mod inutil în mai multe situaţii, mai ales dacă este vorba de o informaţie ce se obţine în urma unor prelucrări mai complexe.

O situaţie de ieşire trebuie să îndeplinească, în general, următoarele condiţii:

- să se refere la unul din obiectivele sistemului;
- să acopere integral domeniul la care se referă;
- să fie cât mai concisă, să elimine detaliile ce nu sunt necesare;
- să fie prezentată într-o formă simplificată şi inteligibilă;
- să apară la timp;
- informaţiile pe care le conţine să fie corelate cu informaţiile din celelalte situaţii.

Conţinutul şi forma situaţiilor de ieşire se stabilesc de către colectivul de proiectare împreună cu beneficiarul sistemului care va viza nominal macheta fiecărei situaţii de ieşire.

Situaţiile de ieşire dintr-un sistem informaţional economic pot fi clasificate după

următoarele criterii:

a) specificul funcţiilor generale ale agentului economic;

b) destinaţie;

c) gradul de sintetizare a indicatorilor;

d) momentul generării;

e) intervalul de referinţă;

f) modul de obţinere.

a) După specificul funcţiilor generale ale agentului economic urmărit în funcţie de natura activităţii de bază situaţiile de ieşire se pot grupa în grupe omogene corespunzătoare funcţiilor agentului economic:

- situaţii de ieşire pentru activitatea de producţie;
- situaţii de ieşire pentru activitatea comercială;
- situaţii de ieşire pentru activitatea financiar-contabilă;
- situaţii de ieşire pentru activitatea de personal.

b) După destinaţie situaţiile de ieşire pot fi:

- situaţii de ieşire destinate altor sisteme;
- situaţii de ieşire destinate sistemului propriu.

Situaţiile de ieşire destinate altor sisteme (agenţi economici sau organisme de stat) au ca scop furnizarea de informaţii specifice cooperării în realizarea activităţilor proprii şi de informare a organismelor de stat. Acestea sunt stabilite prin acte normative sau prin înţelegeri între unităţi, motiv pentru care structura lor este obligatorie pentru proiectant. În această categorie se includ rapoartele statistice ale principalilor indicatori economico-financiari, dările de seamă, situaţiile operative între unităţi referitoare la stadiul relaţiilor contractuale, etc. Tot aici se includ şi documentele primare generate prin prelucrare automată a datelor (factura fiscală, avizul de însoţire a mărfii etc.) care circulă şi în exteriorul sistemului propriu.

Situaţiile de ieşire destinate sistemului propriu al agentului economic cuprind informaţii necesare atât compartimentelor funcţionale, cât şi organelor de conducere ale unităţi. De exemplu: Situaţia realizării producţiei, Situaţia livrărilor pe beneficiari, Situaţia facturilor emise şi încasate, etc.

c) Din punct de vedere al gradului de sintetizare a indicatorilor avem:

- situaţii de ieşire ce conţin informaţii cu caracter sintetic;
- situaţii de ieşire ce conţin informaţii cu caracter analitic.

Situaţiile de ieşire ce conţin informaţii cu caracter sintetic servesc pentru conducerea de ansamblu a agentului economic şi cuprind indicatori globali de urmărire şi control cum ar fi : profitul, cifra de afaceri, productivitatea muncii, consumurile, producţia marfă etc.

Situaţii de ieşire ce conţin informaţii cu caracter analitic descriu în detaliu elementele patrimoniale ale agentului economic, operaţiile economice legate de acestea şi sunt destinate compartimentelor funcţionale pentru realizare unor lucrări curente. De exemplu: Fişa de urmărire a aprovizionării, Situaţia materialelor fără mişcare, Graficul de livrare, Situaţia realizării produc în ziua de .. , etc.

d) Momentul generării asigură obţinerea de situaţii cu caracter:

- operativ;
- periodic;
- aleator.

Situaţiile de ieşire cu caracter operativ sunt elaborate la perioade mai mici (schimb, zi, semidecadă, decadă,etc.) şi conţin informaţii operative cu un grad redus de prelucrare necesare conducerii prin excepţie. De exemplu: Situaţia realizării producţiei în ziua de .., Analiza realizării planului de desfacere în ziua de ..., etc.

Situaţiile de ieşire cu caracter periodic se elaborează la intervale mai mari de timp (lunar, trimestrial, anual) şi conţin informaţii sintetice ce caracterizează evoluţia fenomenelor şi proceselor economice. Sunt destinate anumitor factori de decizie, compartimentelor funcţionale sau altor sisteme şi se caracterizează prin termene fixe de elaborare. De exemplu: Situaţia

facturilor emise şi încasate pe luna ... , Balanţa analitică a materialelor pe luna... , Balanţa sintetică pe luna ..., etc.

Situaţiile de ieşire cu caracter aleator sunt obţinute cu frecvenţe şi la termene nedeterminate şi conţin informaţii sintetice sau analitice ce reflectă aspecte pozitive sau negative din activitatea agentului economic, fiind destinate, de regulă, conducerii de pe treptele superioare (director, consiliu de administraţie, etc.) De exemplu: Situaţia stocurilor ce nu au mişcare de "n" zile, Situaţia facturilor neîncasate de "n" zile, etc.

e) După intervalul de referinţă situaţiile de ieşire se pot structura în trei categorii:

- de stare;
- statistice;
- previzionale.

Situaţiile de ieşire cu caracter de stare conţin informaţii ce evidenţiază nivelul şi disponibilul de resurse la un moment dat, reflectând elemente de patrimoniu ale agentului economic cum ar fi: stocuri de materiale şi produse, facturi în sold, nivelul veniturilor, etc.

Situaţiile de ieşire cu caracter statistic grupează informaţii pe mai multe perioade de gestiune pentru a stabili tendinţele evolutive ale fenomenelor şi proceselor economice în vederea formulării unor aprecieri de ansamblu. Din această categorie fac parte situaţiile transmise Comisiei Naţionale de Statistică, organelor Ministerului Finanţelor sau altor organisme publice.
Situaţiile de ieşire cu caracter previzional redau informaţii ce reflectă tendinţele fenomenelor şi proceselor economice în scopul stabilirii obiectivelor viitoare ale agentului economic.

f) După modul de obţinere situaţiile de ieşire sunt:

- situaţii de ieşire obţinute la imprimantă;
- situaţii de ieşire obţinute la videoterminal.

Situaţii de ieşire obţinute la imprimantă se folosesc de regulă în cazul în care cadrul legislativ prevede utilizarea lor ca documente justificative ce se vor arhiva după consultare. De exemplu: Statele de salarii, Jurnalul privind vânzările, Situaţia centralizatoare a intrărilor de materiale, etc.

Situaţii de ieşire obţinute la videoterminal sunt de regulă situaţii de urmărire operativă a unor indicatori economici şi se folosesc în toate cazurile în care cadrul legislativ nu prevede utilizarea lor ca documente justificative.

În definirea conţinutului şi modului de obţinere a situaţiilor de ieşire proiectantul va ţine seama de cerinţele beneficiarului dar va avea în vedere şi performanţele şi limitele sistemelor electronice de calcul şi ale mediului de programare cu care lucrează.

La definitivarea fiecărei situaţii de ieşire trebuie să se precizeze următoarele elemente:

- antetul situaţiei, care precizează date de identificare a unităţii (subunităţii) beneficiare, data şi eventual codul situaţiei;
- titlul situaţiei, care redă într-o formă sintetică conţinutul informaţional al acesteia şi perioada de referinţă;
- capul de tabel, respectiv prezentarea indicatorilor din fiecare coloană urmărindu-se gruparea logică a acestora : indicatori informativi (de exemplu cod, denumire), indicatori de grupare (de exemplu unitate de măsură, preţ) şi indicatori cantitativ-valorici;
- natura şi lungimea maximă a fiecărui indicator;
- benzile de total;
- numărul de exemplare;
- destinaţia fiecărui exemplar;
- periodicitatea şi termenele de obţinere;
- dispozitivul periferic de obţinere.

4.4.3. Proiectarea bazei informaţionale de intrare

Proiectarea bazei informaţionale de intrare înseamnă determinarea completă şi corectă a mulţimii atributelor de intrare necesare şi suficiente pentru obţinerea ieşirilor sistemului

informatic şi gruparea acestora într-un ansamblu omogen de entităţi în vederea construirii în proiectarea de detaliu a fişierelor sau a bazelor de date.

Proiectarea bazei informaţionale de intrare are două subfaze:

a) determinarea conţinutului bazei informaţionale de intrare şi a algoritmilor folosiţi;
b) structurarea bazei informaţionale în entităţi.

a) Determinarea conţinutului bazei informaţionale de intrare.

Pentru a stabili necesarul atributelor de intrare trebuie examinat modul în care are loc obţinerea informaţiilor cuprinse în situaţiile de ieşire proiectate sau în documentele inventariate în sistem (în funcţie de varianta de lucru aleasă în proiectarea generală).

Din acest punct de vedere se disting două categorii de informaţii:

- informaţii ce se obţin în urma aplicării unui algoritm de calcul (de exemplu: stocul unui material, soldul unui cont, numărul de zile restanţă la plata unei facturi, etc.)
- informaţii obţinute prin simpla transferare a unor valori similare aflate la intrare (de exemplu: codul unui produs, denumirea unui produs, numele unei persoane, etc.)

Informaţiile din prima categorie trebuie descompuse pe baza algoritmului de calcul în operanzi primari ai acestora. Apoi aceştia împreună cu informaţiile preluate (transferate) din a doua categorie reprezintă elementele necesare obţinerii informaţiilor solicitate.

Deci, ansamblul atributelor de intrare este reprezentat de totalitatea informaţilor obţinute prin preluare şi a operanzilor primari după reducerea elementelor comune.

Se poate defini următorul procedeu de determinare a bazei informaţionale de intrare:

- se inventariază câmp cu câmp informaţiile cuprinse în situaţiile de ieşire proiectate;
- dacă informaţiile rezultă dintr-un calcul se identifică algoritmul, apoi operanzii primari se introduc în baza informaţională de intrare dacă nu au fost deja incluşi;
- dacă informaţia este transferată se include în baza informaţională de intrare dacă nu a fost inclusă deja.

Consistenţa şi relevanţa bazei informaţionale de intrare sunt dependente de acurateţea algoritmilor în corespondenţă cu specificul sistemului obiect.

Ansamblul atributelor de intrare va fi apoi analizat şi structurat în entităţi, iar algoritmii de calcul vor fi descrişi în procedurile de exploatare. Algoritmii determinaţi se pot exprima ca expresii algebrice, liste de acţiuni condiţionale, tabele de decizii.

b) Structurarea bazei informaţionale de intrare în entităţi.

Cea de a doua subfază presupune separarea ansamblului atributelor de intrare în grupe omogene şi stabilirea legăturilor dintre acestea. Pentru aceasta se rezolvă următoarele probleme:

- se analizează baza informaţională de intrare determinată anterior;
- se grupează atributele în entităţi.

Analiza bazei informaţionale de intrare are scopul de a determina modul de participare a datelor la procesul de prelucrare automată. Pentru a efectua această analiză se au în vedere elementele funcţionale indivizibile ale unui sistem informatic, respectiv entităţile. Entitatea poate fi privită ca un aspect al existenţei delimitat ca întindere, conţinut şi sens şi desemnează un element conceptual sau material indivizibil din punct de vedere funcţional. Ea este deci o componentă structural-funcţională elementară a sistemului care are o existenţă bine determinată şi reprezintă o individualitate. De exemplu: entitatea personal, entitatea materiale, entitatea costuri, etc. Entitatea este caracterizată printr-o mulţime de atribute preluate din nucleul informaţional în urma aplicării unor criterii de structurare. De exemplu entitatea personal are ca atribute: numele persoanei, prenumele persoanei, sexul, data naşterii, etc.

Pentru a realiza analiza corectă a bazei informaţionale de intrare este necesară distincţia clară între atribut şi valoarea pe care o poate avea. De exemplu NUME este un atribut (un nume de dată, o caracteristică) a entităţii PERSONAL iar Popa, Sandu, Radu, etc. sunt valori pe care le ia atributul respectiv. Din perspectiva proiectării şi realizării unui sistem informatic interesează identificarea atributului şi nu a valorilor concrete pe care acesta le poate lua.

Valoarea informaţională a unei entităţi cuprinde trei aspecte:

- caracteristici determinante specifice care identifică entitatea şi o individualizează în ansamblul sistemului;
- nivelul sau starea entităţii la un moment dat;
- transformările sau schimbările care intervin datorită funcţiilor sistemului şi determină trecerea de la o stare la alta.

Poziţia unui atribut în raport cu o entitate îi conferă acestuia un mod specific de comportare şi de aici rezultă trei feluri de atribute: cu caracter permanent, variabil şi de stare.

Atributele permanente rămân neschimbate o perioadă mai mare de timp, se preiau în sistem o singură dată şi se actualizează la intervale mai mari de timp. Ele intră în componenţa entităţilor permanente. De exemplu: marca, nume, sex, etc. sunt atribute permanente ale entităţii de personal.

Atributele variabile se caracterizează prin aceea că valoarea lor variază de la o prelucrare la alta reflectând fenomene şi procese economice ce s-au desfăşurat de la prelucrarea precedentă. Ele sunt introduse în sistem la fiecare prelucrare şi intră în componenţa entităţilor variabile. De exemplu: timpul lucrat, producţia realizată de o persoană, indicele individual de acord, etc. sunt atribute variabile ale entităţii de personal.

Atributele de stare caracterizează nivelul unui element la un moment dat şi reprezintă rezultatul unei prelucrări. Ele sunt stocate în baza informaţională de unde se preiau periodic, se prelucrează şi se înregistrează la noua valoare. Ele nu reflectă fenomene şi procese economice ce au avut loc între două prelucrări, deci sunt diferite de atributele variabile, dar nu au nici o valoare constantă, ci ea se schimbă la fiecare prelucrare şi, deci, nu sunt nici atribute permanente. Ele se introduc în entităţi permanente, iar când sunt mai numeroase se constituie în entităţi de stare. De exemplu: vârsta, numărul de copii, suma câştigurilor de la începutul anului, etc. sunt atribute de stare ale entităţii de personal.

Pe baza acestei abordări poate fi definit modelul conceptual de prelucrare pe calculator la fiecare ciclu astfel:

- se actualizează atributele permanente cu eventualele modificări intervenite;
- se introduc în sistem valorile atributelor variabile prin preluarea din documente justificative a datelor ce reflectă operaţii economice efectuate de la prelucrarea anterioară;
- se actualizează atributele de stare pe baza valorii iniţiale şi a mişcărilor reflectate de atributele variabile, se calculează starea finală pentru fiecare atribut de stare;
- se asociază şi se prelucrează atributele permanente, variabile şi de stare pentru obţinerea situaţiilor de ieşire proiectate.

Gruparea atributelor în entităţi are la bază analiza efectuată anterior şi constă în separarea ansamblului unic de date în grupuri omogene în funcţie de tipul atributelor, de aici rezultă că şi entităţile sunt permanente, variabile şi de stare.

Relaţiile dintre entităţi pot fi:

- atributele din entităţi permanente se actualizează pe baza tranzacţiilor externe ce reflectă modificări în structura şi componenţa acestora;
- entităţile variabile se creează la fiecare prelucrare pe baza tranzacţiilor externe ce reflectă operaţiile economice;
- entităţile de stare se actualizează pe baza tranzacţiilor interne.

Gruparea datelor în entităţi permanente se face în funcţie de următoarele criterii:

- atribute de identificare;
- frecvenţa de utilizare şi actualizare;
- accesul la date.

Primul criteriu impune gruparea în aceeaşi entitate a tuturor atributelor ce au un element de identificare comun. De exemplu salariul, numărul de copii, data naşterii, adresa se pot grupa într-o singură entitate având comun elementul de identificare marca sau numele.

Al doilea criteriu determină gruparea în aceeaşi entitate a atributelor care au aceeaşi frecvenţă de utilizare şi actualizare. De exemplu sporul de noapte, sporul şantier, sporul pentru

condiţii deosebite se pot grupa într-o altă entitate decât cea anterioară (deşi se referă la aceeaşi persoană) datorită criteriului precizat.

Al treilea criteriu se referă la gruparea în aceeaşi entitate a atributelor pentru care sunt formulate cerinţe deosebite legate de consultarea bazelor de date şi are importanţă deosebită în sistemele interactive la care este necesar să se asigure un timp de răspuns cât mai redus.

Gruparea datelor în entităţi variabile se face avându-se în vedere posibilitatea grupării sub un identificator comun şi în funcţie de omogenitatea datelor din punct de vedere al locului şi momentului preluării lor. Aceasta impune gruparea în aceeaşi entitate a tuturor atributelor ce se regăsesc pe acelaşi document de mişcare.

În cazul în care numărul de documente este relativ mare, iar ponderea lor este inegală în sistem se poate apela la gruparea atributelor în funcţie de operaţiile economice pe care le reflectă. De exemplu ieşirile de produse se reflectă în mai multe documente: factura fiscală, avizul de însoţire a mărfii, lista de inventariere, etc. În măsura în care ieşirile prin vânzare au ponderea cea mai mare se poate apela la gruparea atributelor ce reflectă toate ieşirile într-o singură entitate, codificându-se tipurile de ieşiri.

Gruparea atributelor în entităţi de stare se face în funcţie de aceleaşi criterii ca şi cele avute în vedere la entităţile permanente. De regulă, numărul atributelor de stare este mai mic şi soluţia cea mai avantajoasă este aceea a grupării lor în entităţile permanente la care se referă, obţinându-se astfel entităţi cu un grad mai ridicat de integrare. De exemplu atributele stoc şi data ultimei mişcări pot fi înregistrate în entitatea permanentă produs care va oferi astfel o imagine mai completă asupra unui produs la un moment dat.

Constituirea entităţilor în cadrul bazei informaţionale de intrare poate fi privită ca o operaţie de separare a ansamblului unic de date în mai multe grupe omogene. Aceste grupe ar trebui, la prima vedere, să constituie submulţimi disjuncte pentru a evita preluările repetate. Dar, în cursul prelucrării, va fi necesară reconstituirea parţială a ansamblului iniţial de atribute, respectiv cuplarea a două sau mai multe entităţi . Pentru aceasta entităţile ce se vor cupla trebuie să aibă atribute comune pentru a permite realizarea asocierilor necesare.

Atributele prezente simultan în două sau mai multe entităţi se numesc atribute de legătură. Ansamblul atributelor de legătură dintre entităţi se determină pe baza asocierilor pe care le impun informaţiile ce vor trebui obţinute în situaţiile de ieşire şi algoritmii de calcul.

Trebuie menţionat că nu toate atributele ce se pot găsi simultan în mai multe baze de date pot constitui atribute de legătură. Pentru aceasta atributul respectiv trebuie să constituie identificator pentru cel puţin una din entităţile asociate. De exemplu pentru obţinerea unei situaţii de ieşire utilizăm entităţile cu structura de mai jos:

- entitatea produs cu atributele: cod produs, denumire, unitate de măsură, preţ;
- entitatea mişcări cu atributele: cod produs, preţ, cantitate.

Atributul preţ, deşi este prezent în ambele entităţi, nu poate servi drept atribut de legătură deoarece nu identifică în mod unic nici una din entităţile ce cuplează. De aceea se va folosi ca atribut de legătură codul produsului.

4.4.4. Proiectarea codurilor

O cerinţă de bază a sistemelor informatice este aceea ca elementele din sistem să fie riguros definite, ordonate şi clasificate prin codificare.

Codificarea reprezintă o activitate cu implicaţii mari în prelucrarea automată a datelor regăsindu-se sub două aspecte :

- pe de o parte, ca o codificare internă a datelor în calculatorul electronic;
- pe de altă parte, ca o codificare externă, ceea ce permite introducerea în prelucrare a unor date cât mai împachetate, mai formalizate.

Prin codificare înţelegem generarea unor grupuri de simboluri, denumite coduri şi repartizarea acestora valorilor concrete pe care le iau elementele diferitelor mulţimi (în cazul unui sistem informatic se codifică atât atribute din baza de date cât şi elementele structurale şi funcţionale ale sistemului)

Rezultatul activităţii de codificare este concretizat într-un sistem de coduri.

Prin cod se înţelege o combinaţie de simboluri asociate unui ansamblu de date. Între mulţimea simbolurilor ce urmează a fi codificate şi mulţimea codurilor asociate se stabileşte o corespondenţă biunivocă.

Totalitatea combinaţiilor distincte, posibile de realizat din simbolurile ce compun un cod reprezintă capacitatea unui sistem de coduri.

Un cod trebuie să aibă o capacitate acoperitoare pentru toate situaţiile posibile cu condiţia păstrării unicităţii codului.

Capacitate unui sistem de coduri numeric se determină cu formula:

$$C = b^n - 1$$

unde : C = capacitatea codului
b = baza sistemului de numeraţie utilizat
n = numărul de poziţii numerice din cadrul codului

Capacitatea unui sistem de coduri alfanumeric se determină cu formula:

$$C = 24^a * b^n$$

unde : C = capacitatea codului
a = numărul de poziţii alfabetice din cadrul codului
b = baza sistemului de numeraţie utilizat
n= numărul de poziţii numerice din cadrul codului

Numărul de simboluri elementare dintr-un cod poartă denumirea de lungime a secvenţei de cod (lungimea codului).

Lungimea codului trebuie să fie minimă pentru a reduce timpul de transmisie şi preluare a datelor pe suporţii tehnici de date, pentru a reduce timpul de prelucrare etc.

În acelaşi timp, lungimea codului trebuie să-i asigure şi o capacitate corespunzătoare. Aceasta se poate estima cu relaţia: $L >= \log_K N$

unde: L = lungimea codului
N = numărul de elemente ce se codifică
K = numărul de simboluri utilizate în construirea codului

Forma finală a codului cu precizarea clară a numărului de poziţii utilizate, natura acestora, cifra de control şi algoritmul de calcul al acesteia reprezintă formatul codului.

În proiectarea unui sistem de coduri trebuie să avem în vedere două aspecte importante şi anume:

- influenţa tipului şi structurii codului asupra performanţelor operaţiilor de prelucrare automată a datelor din sistemul informatic;
- implicaţiile utilizării codurilor în operaţiile de culegere şi preluare a datelor şi de interpretare a rezultatelor finale de către utilizatorii neinformaticieni.

Având în vedere aceste considerente, se impune ca în proiectarea unui sistem de coduri să se respecte o serie de cerinţe:

1. **Unicitate.** Fiecărui element din mulţimea codificată i se atribuie un cod unic, această cerinţă trebuind să fie asigurată la nivelul întregului sistem informatic.
2. **Stabilitate.** Caracteristica codificată trebuie să rămână neschimbată o perioadă cât mai mare de timp.
3. **Elasticitate.** Să permită inserări şi extensii ale nomenclatoarelor de coduri în vederea includerii de noi elemente.
4. **Conciziune.** Să se utilizeze un număr cât mai mic de simboluri în construirea unui cod.
5. **Claritate.** Să permită realizarea cu uşurinţă a operaţiunilor de codificare-decodificare.
6. **Semnificaţie.** Să permită, pe cât posibil, sugerarea caracteristicilor codificate şi să genereze interes pentru a facilita utilizarea codurilor.
7. Codul să fie **operaţional.** Să permită prelucrarea automată a datelor şi efectuarea cu uşurinţă a operaţiilor de sortare, indexare, scindare, interclasare etc.

Un sistem de coduri are următoarele funcţii:

a) **Funcţia de caracterizare**, care asigură exprimarea într-o formă concisă, unică şi stabilă în timp, a conţinutului semantic al fiecărui element codificat prin intermediul codului asociat acestuia. În mod concret funcţia de caracterizare permite utilizarea cu prioritate a codului în locul denumirii integrale a elementului codificat.

b) **Funcţia de identificare**, care oferă posibilitatea regăsirii mai rapide a elementelor prin intermediul codurilor asociate lor decât prin folosirea completă a semanticii acestora. Această funcţie creează posibilitatea ulterioară de selectare a anumitor caracteristici prin intermediul cărora se vor identifica în mod unic valori folosind conceptul de cheie de acces.

c) **Funcţia de control**, care presupune existenţa unei chei de control (formată din unul sau mai multe caractere) pe baza căreia folosind metode şi algoritmi specifici să se poată verifica integral corectitudinea simbolurilor care intră în structura codului. De regulă, această cheie de control se plasează în ultima parte a codului şi este separată de acesta printr-o linie de unire.

d) **Funcţia de manipulare** a elementelor codificate, care facilitează introducerea în memorie a acestora, reducerea timpului de prelucrare, inclusiv uşurinţa folosirii codului de către personalul din compartimentele funcţionale implicate în funcţionarea sistemului informatic.

Tipuri de coduri. Diversitatea şi complexitatea colecţiilor de date, specificul operaţiilor de regăsire, sortare, indexare etc. au condus la apariţia unei palete variate de coduri, ce se pot grupa după mai multe criterii .

I). După natura caracterelor utilizate:

1. Coduri numerice, în care simbolurile utilizate în construirea codului sunt cifrele de la 0 la 9. De exemplu codificând secţiile din facultate: 11 pentru Management; 12 pentru Contabilitate; 13 pentru Informatică etc.

2. Coduri alfabetice, în care simbolurile utilizate în construirea codului sunt literele alfabetului. De exemplu : MIZ pentru Management, cursuri de zi; CIS pentru Cibernetică, informatică şi statistică etc.

3. Coduri alfanumerice, în care simbolurile utilizate sunt cifrele, literele şi toate caracterele speciale cuprinse în codul ASCI.

II. După lungimea secvenţei de cod există:

1. Coduri cu lungime fixă, în care toate elementele unei mulţimi sunt codificate cu acelaşi număr de caractere.

2. Coduri cu lungime variabilă, în care lungimea secvenţei de cod poate fi diferită pentru elemente diferite din aceeaşi mulţime.

III. După semnificaţia codului sunt:

1. Coduri semnificative. Aceste coduri semnifică conţinutul informaţional al elementelor pe care le reprezintă. De exemplu: lect - semnifică valoarea lector, conf - semnifică valoarea conferenţiar, stud - semnifică student etc.

2. Coduri nesemnificative. Aceste coduri nu semnifică conţinutul informaţional al elementelor pe care le reprezintă. Ele sunt combinaţii de simboluri după anumite criterii şi se atribuie elementelor fără a avea o semnificaţie. De exemplu: 01 - pentru lector, 02 - pentru conferenţiar etc.

IV. După modul de întocmire a nomenclatoarelor de coduri există:

1. *Coduri atribuite manual*;
2. *Coduri atribuite automat.*

V. Din punct de vedere al utilizatorului se pot folosi:

1. *Coduri accesibile utilizatorului;*
2. *Coduri ascunse utilizatorului.*

VI. După structura codului există:

1. *Coduri elementare;*
2. *Coduri complexe.*

Codurile elementare au rolul de a identifica un element din cadrul unei singure mulţimi de elemente. Ele pot fi:

- coduri secvenţiale,
- coduri secvenţiale cu formare de grupe,
- coduri abreviate.

Codurile secvenţiale se construiesc prin atribuirea în ordine crescătoare a unor simboluri numerice sau alfanumerice elementelor din mulţimea ce urmează a fi codificată, pe măsură ce aceste elemente apar în sistem. Ele pot avea lungim fixă sau variabilă. În cazul codurilor cu lungime fixă se adaugă zerouri nesemnificative în faţa şirurilor de caractere atribuite.

Codurile secvenţiale au avantajul conciziunii, dar nu sunt elastice şi nu pot fi utilizate eficient în gruparea datelor.

Codurile secvenţiale cu formare de grupe. Reprezintă o dezvoltare a codurilor secvenţiale, în sensul că se prevede rezervarea unor grupe de coduri pentru grupe de elemente, iar în cadrul grupelor elementele sunt codificate secvenţial.

Separarea mulţimii elementelor de codificat pe grupe sau subgrupe se face pe baza unor caracteristici comune tuturor elementelor ce aparţin grupei sau subgrupei respective.

Acest tip de coduri este elastic, servind şi necesităţilor de grupare şi ordonare a datelor.

Coduri abreviate. Sunt coduri alfabetice, de lungime fixă sau variabilă care se constituie prin prescurtare sau prin preluarea unor iniţiale.

Cele obţinute prin prescurtare poartă denumirea de *mnemonice.*

Cele obţinute prin preluarea unor iniţiale poartă denumirea de *acronime*.

Codurile complexe. Se folosesc pentru elementele ce pot să aparţină mai multor mulţimi distincte care pot fi folosite în comun pentru prelucrări viitoare. Aceste coduri sunt astfel construite încât să reflecte apartenenţa multiplă. El pot fi:

- coduri ierarhizate,
- coduri juxtapuse,
- coduri combinate.

Codurile ierarhizate se folosesc atunci când între caracteristicile ce urmează a fi simbolizate (codificate) sunt stabilite relaţii de subordonare.

Codurile juxtapuse (partajate) se construiesc prin concatenarea codurilor atribuite caracteristicilor individuale cu semnificaţii distincte.

Codurile combinate surprind în structura lor relaţiile logice existente între diferite caracteristici ale elementelor de codificat. Ele pot fi matriciale sau binare.

Codurile matriciale se folosesc în cazul în care elementele de codificat pot fi caracterizate în funcţie de două însuşiri, codul atribuit reprezentându-le cumulativ pe amândouă.

Codurile binare se folosesc pentru a reflecta în mod sugestiv şi simplu prezenţa sau absenţa mai multor însuşiri ce caracterizează simultan elementele unei mulţimi de date. În acest sens, însuşirile respective se codifică prin puteri succesive ale cifrei doi, fiecare poziţie putând lua valoarea 0 sau 1 în funcţie de prezenţa sau absenţa însuşirii respective.

VII. După modul de preluare în sistemul de calcul, codurile pot fi:

1. *Coduri preluate manual* (prin tastare de la terminal);
2. *Coduri preluate automat*. Ele sunt introduse în sistem prin utilizarea unor echipamente periferice speciale precum: echipamente pentru citirea documentelor scrise cu cerneală magnetică, scannere, camere video şi alte echipamente multimedia.

VIII. După modul în care tratează erorile provenite în urma utilizării unui sistem de coduri întâlnim:

1. *Coduri care nu detectează erorile de manipulare;*
2. *Coduri autodetectoare de erori;*
3. *Coduri autocorectoare de erori.*

Activităţile parcurse în elaborarea unui sistem de coduri sunt:

1. analiza elementelor ce urmează a fi codificate;
2. precizarea şi uniformizarea terminologiilor;
3. stabilirea caracteristicilor şi a relaţiilor dintre elementele de codificat;
4. alegerea tipurilor de coduri;

5. estimarea capacităţii, lungimii şi formatului codurilor;
6. determinarea cifrei de control corespunzătoare fiecărui cod şi asocierea ei codului respectiv;
7. atribuirea codurilor elementelor de codificat, respectiv crearea nomenclatoarelor de coduri;
8. întreţinerea nomenclatoarelor de coduri.

O problemă deosebită care se ridică în legătură cu codificarea datelor este acea a verificării corectitudinii codului în procesele de culegere, transmitere şi prelucrare a datelor.

Cu ocazia manipulării codului, apar posibilităţi ca acesta să fie modificat, erorile astfel generate având consecinţe negative asupra corectitudinii informaţiilor obţinute şi asupra funcţionării sistemului informatic în ansamblu.

Pe parcursul exploatării sistemului informatic pot să apară erori datorate:

- transpunerii greşite pe documentul justificativ sau pe elementul pe care codul îl identifică;
- preluării incorecte (de pe documentele justificative de exemplu) în sistemul de calcul.

Se pune deci problema de a utiliza metode ce pot prevenii sau depista erorile în codificare şi în utilizarea codurilor. În acest sens, o tehnică des utilizată este cea a cheii (cifrei) de control.

4.5. PROIECTAREA DE DETALIU

4.5.1. Caracteristicile generale ale proiectării de detaliu

Obiectivul fundamental al proiectării de detaliu constă în realizarea cerinţelor funcţionale definite în proiectarea generală, prin transformarea modelului conceptual într-un model tehnic, operaţional. În acest scop se alege soluţia optimă de gestiune a datelor, bazată pe principiul utilizării fişierelor sau a bazelor de date şi se proiectează structurile de date, inclusiv prelucrările specifice la nivelul subsistemelor (sau al sistemului informatic în ansamblu) şi prelucrările specifice de la nivelul procedurilor.

Proiectarea de detaliu are un caracter preponderent tehnic, iar activităţile desfăşurate sunt influenţate direct de soluţia de gestiune a datelor aleasă, de caracteristicile sistemului electronic de calcul, de sistemul de operare, precum şi de mediul de programare ce va fi utilizat. În realizarea practică a proiectării de detaliu a sistemului informatic se parcurg următoarele faze: proiectarea fişierelor; stabilirea ordinii de prelucrare a fişierelor; determinarea procedurilor; proiectarea procedurilor.

4.5.2. Proiectarea fişierelor

Urmăreşte determinarea celei mai adecvate structuri de date pentru entităţile bazei informaţionale anterior stabilite, în vederea optimizării obţinerii ieşirilor proiectate. Pentru aceasta trebuie realizate următoarele activităţi:

- definitivarea dicţionarului de date şi a clasificării fişierelor de bază,
- proiectarea structurii înregistrărilor,
- alegerea modului de organizare a fişierelor de bază şi a suporturilor de memorie externă,
- definirea măsurilor de protecţie şi securitate.

4.5.3. Stabilirea ordinii de prelucrare a fişierelor de bază

Ordinea de prelucrare a fişierelor de bază este determinată de restricţiile de integritate dintre fişiere, determinate ca reguli sau condiţii formulate în scopul menţinerii coerenţei datelor în timpul operaţiunilor de prelucrare.

Aceste restricţii sunt evidenţiate prin subordonarea dintre fişiere, care impun ca valoarea unuia sau mai multor câmpuri dintr-un fişier să se regăsească în calitate de cheie primară sau externă pentru alt fişier. Pentru a exemplifica, să considerăm fişierul PRODUS (un nomenclator ce conţine date permanente şi de stare) şi MIŞCĂRI (un fişier cu date variabile ce conţine intrările şi ieşirile de produse). Fiecare produs ce se găseşte în fişierul MIŞCĂRI trebuie sa aibă un corespondent în fişierul PRODUS, câmpul COD_PROD fiind în acelaşi timp cheie primară pentru fişierul PRODUS şi cheie externă pentru fişierul MIŞCĂRI.

Restricţiile de integritate dintre fişiere determină ordinea de prelucrare a lor, astfel încât crearea şi actualizarea unui fişier să se facă numai după ce s-a efectuat actualizarea tuturor fişierelor de care acesta depinde. Regula este că se actualizează în primul rând fişierele cu date permanente (nomenclatoarele) şi apoi fişierele variabile. În acest context fişierul PRODUS trebuie actualizat înaintea prelucrării fişierului MIŞCĂRI, pentru a putea reflecta şi mişcările referitoare la produsele noi.

Restricţiile de integritate dintre fişiere pot fi privite în sens static şi în sens dinamic. În sens static acestea reflectă, după cum am văzut, modul de subordonare a prelucrărilor unui fişier în raport cu altul, ceea ce condiţionează în mod direct ordinea de prelucrare. Restricţiile de integritate dintre fişiere, abordate în sens dinamic exprimă reguli sau corelaţii existente înainte sau pe parcursul prelucrării acestora şi sunt, în general, dictate de particularităţile sistemului obiect. De exemplu, în cadrul operaţiilor privind mişcările de produse vor fi prelucrate mai întâi intrările şi apoi ieşirile, pentru a evita apariţia de stocuri intermediare negative (anomalie din punct de vedere economic).

Deci într-o aplicaţie informatică economică ordinea de apelare a procedurilor trebuie să fie următoarea: proceduri de actualizare a fişierelor permanente; proceduri de actualizare a fişierelor variabile; proceduri de actualizare a datelor (fişierelor) de stare; proceduri de obţinere a ieşirilor.

4.5.4. Determinarea procedurilor

Într-un sistem informatic economic procedura poate fi definită ca o secvenţă de operaţii repetitive executate fără întreruperi externe de la acelaşi post de lucru, care transformă un anumit ansamblu de intrări în anumite ieşiri, după reguli bine definite.

Pentru fiecare procedură va trebui să specificăm următoarele elemente: funcţia procedurii; intrările în procedură; ieşirile din procedură; logica internă de prelucrare; interfeţele cu alte proceduri.

Funcţia procedurii indică natura prelucrărilor sale asupra fişierelor, existând mai multe tipuri: proceduri pentru dirijarea altor proceduri (aşa numitele proceduri de comandă); proceduri pentru introducerea şi validarea datelor; proceduri pentru actualizarea fişierelor; proceduri pentru exploatarea fişierelor şi obţinerea ieşirilor; proceduri pentru reorganizarea fişierelor de bază; proceduri pentru protecţia şi securitatea fişierelor de bază.

Intrările în proceduri sunt constituite din parametri de apel şi date de intrare din fişierele de bază. Fiecare tip de intrare trebuie descris, în documentaţie, în forma standard, respectiv cu specificatorul complet de fişier. Ieşirile din proceduri sunt constituite din fişierele create sau situaţiile de ieşire obţinute şi trebuie descrise, ca şi intrările, cu specificatorul complet de fişier.

Logica internă de prelucrare indică modul concret de transformare a intrărilor în ieşiri (validări, conversii, actualizări, ordonări, calcule etc.). Ea se poate prezenta în proiect cu orice metodă de descriere a algoritmilor: scheme logice, tabele de decizii, limbaj pseudocod etc.

Interfeţele dintre proceduri sunt constituite din ansambluri de date (fişiere) create intr-o procedură la momentul "t" care pot constitui intrări pentru aceeaşi procedură la momentul "t+1" sau pentru o altă procedura la momentul "t" sau "t+1".

Determinarea procedurilor se face în funcţie de anumite criterii dintre care menţionăm:

- tipologia şi natura prelucrărilor din subsistem (actualizări, exploatări etc.);
- frecvenţa şi termenele de realizare a prelucrărilor fişierelor de bază sau a obţinerii situaţiilor de ieşire;
- natura şi specificul intrărilor şi ieşirilor din subsistem;

- ordinea de prelucrare a fişierelor de bază;
- necesitatea asigurării unor proceduri de dirijare;
- asigurarea unor interfeţe optime intre proceduri etc.

Este indicat ca în proiectarea procedurilor să se folosească metoda top-down, pornindu-se de la cele două funcţii principale pe care trebuie să le asigure un sistem informatic economic, respectiv actualizarea fişierelor de bază şi obţinerea ieşirilor proiectate.

4.5.5. Proiectarea procedurilor

Are ca obiective definirea tuturor elementelor tehnice necesare elaborării programelor, inclusiv redarea, testarea şi corectarea programelor propriu-zise.

Proiectarea de detaliu a procedurilor (a programelor de aplicaţie) cuprinde:

- proiectarea datelor specifice procedurilor;
- proiectarea prelucrărilor specifice procedurilor.

1. Proiectarea datelor specifice procedurilor. Datele necesare procedurilor sunt concretizate în fişierele de bază ale sistemului proiectat sau în fişierele intermediare impuse de necesitatea detalierii tehnice a realizării funcţiei complexe a fiecărei proceduri.

Proiectarea datelor specifice procedurilor se referă la:

a) intrările în proceduri;

b) ieşirile din proceduri;

c) fişierele intermediare.

a) *Proiectarea intrărilor specifice procedurilor* are în vedere datele de intrare necesare constituirii fişierelor de bază precum şi dialogul procedurii cu utilizatorul, în cazul sistemelor interactive.

Datele de intrare se preiau din documentele de intrare al căror format a fost definitivat integral în proiectarea generala.

Dialogul cu utilizatorul se realizează în următoarele forme:

- prin secvenţe de întrebări şi răspunsuri dirijate de modulele procedurii;
- prin afişarea pe ecran a unei liste de funcţii sau opţiuni de prelucrare din care utilizatorul o selectează pe cea dorită (aşa numita tehnică a meniurilor).

b) *Proiectarea ieşirilor specifice procedurilor* se referă la listele (situaţiile de ieşire) solicitate de beneficiar şi rezultatele controlului fişierelor de bază.

Elementele necesare proiectării ieşirilor au fost definitivate în cadrul proiectării generale, urmând ca în proiectarea de detaliu să se stabilească modul de obţinere a lor, prin programe precizate, precum şi modul de dispunere în pagină sau pe monitor. Datele care formează conţinutul listelor trebuie afişate în aşa fel încât să fie cât mai uşor de consultat, evitând repetarea datelor comune. Aranjarea în pagină a situaţiilor de ieşire trebuie să ia în considerare restricţiile impuse de suportul utilizat, respectiv lungimea maximă a rândului şi numărul de rânduri pe pagină sau pe ecran. O soluţie eficientă de obţinere a situaţiilor de ieşire o reprezintă utilizarea generatoarelor de rapoarte. Un generator de rapoarte este o componentă specializată a unui mediu de programare care creează rezultatele destinate afişării sau imprimării pe baza simplei lor descrieri, fără eforturi de programare.

c) *Proiectarea fişierelor intermediare* se referă la colecţiile tampon de date, diferite de cele din fişierele de bază ale sistemului, utilizate în prelucrare. Ele pot fi:

- fişiere de interfaţă, ce asigură comunicarea între proceduri şi au structuri determinate de setul de date transmise de la o procedură la alta;
- fişiere tampon, ce asigură realizarea anumitor faze ale prelucrării, cum ar fi: sortările, indexările, asocierile etc.

2. Proiectarea prelucrărilor specifice procedurilor. Se referă la:

a) determinarea modulelor de prelucrare;

b) introducerea şi validarea datelor;

c) prelucrarea fişierelor;

d) obţinerea situaţiilor de ieşire.

a) *Determinarea modulelor de prelucrare* urmăreşte descompunerea funcţiei complexe de prelucrare a procedurii în funcţii elementare care să asigure definirea şi executarea succesivă a intrărilor prelucrărilor şi ieşirilor specifice procedurilor.

Modulul asigură anumite prelucrări omogene, are un nume unic şi este reprezentat de o secvenţă finită de instrucţiuni sau comenzi de prelucrare.

Proiectarea procedurilor după metoda programării modulare constă în descompunerea procedurilor în module interconectate în conformitate cu o ierarhie bine precizată. Tehnica modularizării trebuie utilizată, atât la nivelul proiectării şi realizării sistemelor informatice, cât şi la nivelul fiecărei proceduri în parte. Ea presupune identificarea funcţiilor elementare ale procedurii printr-o analiză de sus în jos (top-down) şi realizarea legăturilor de structura dintre module. În cadrul unei proceduri se pot distinge mai multe tipuri de module:

- module de dirijare (module de tip monitor) care coordonează la nivelul procedurii acţiunea altor module; un modul este de tip monitor dacă în structura lui există cel puţin o operaţie de lansare în execuţie a altui modul; un modul monitor poate să conţină numai operaţii de lansare în execuţie a modulelor de nivel imediat inferior, sau uneori şi operaţii care se regăsesc ca atare în algoritmul de prelucrare al procedurii respective;
- module de prelucrare (module operaţionale sau module funcţii) care execută operaţii de prelucrare conform algoritmului precizat;
- module comune folosite de mai multe module de nivel superior sau chiar de proceduri diferite;
- module nefuncţionale care pot să cuprindă descrierea centralizată a datelor, comentarii etc.

b) *Introducerea şi validarea datelor* asigură transpunerea datelor din documentele de intrare anterior proiectate pe un suport tehnic de date şi validarea acestora.

Validarea are rolul de a controla datele de intrare pe baza unui set de reguli predeterminate, prin intermediul unor operaţii de control efectuate de programe sau module de validare. Operaţiile de control pot avea caracter sintactic când se verifică forma, sau semantic când se verifică fondul.

Operaţiile de control sintactic pot să asigure verificarea tipului datelor, a lungimii acestora, pot să controleze cheile de control pentru codurile autodetectoare şi autocorectoare de erori etc.

Operaţiile de control semantic pot fi:

- generale, când urmăresc depistarea unor valori incompatibile (de exemplu: luna 13, ziua 32 etc.);
- de gestiune, când urmăresc depistarea datelor care nu respectă reguli sau restricţii specifice sistemului obiect (de exemplu, în cazul aplicaţiilor economice: verificarea valorii şi duratei de funcţionare pentru ca un obiect sa fie înregistrat ca mijloc fix, controlul stocurilor etc.).

Controlul semantic mai poate fi:

- local când acţionează la nivelul înregistrării (de exemplu: verificarea prezenţei datelor strict necesare intr-o înregistrare; verificarea valorii câmpurilor (16>=vârsta_salariat<=65); verificarea compatibilităţii între valorile a două sau mai multe câmpuri (vârsta şi vechimea în munca, vârsta şi numărul de copii etc.)
- global, când acţionează la nivelul fişierului (de exemplu: verificarea unicităţii cheilor primare; verificarea numărului de înregistrări etc.).

Operaţiile de validare pot fi incluse în proceduri distincte sau pot constitui module în diferite proceduri.

c) *Prelucrarea fişierelor* grupează următoarele operaţii:

- generarea: crearea structurii fişierului;
- iniţializarea fişierului: încărcarea cu date în momentul implementării;
- actualizarea fişierului: ţinerea la zi a fişierului în raport cu schimbările din sistemul obiect; pot fi actualizări simple (interactive, cu date introduse de la terminal) sau complexe (prin prelucrarea datelor înregistrate în alte fişiere)

- exploatarea (consultarea) fişierului pentru obţinerea ieşirilor proiectate;
- reorganizarea fişierului prin modificarea structurii.

d) *Obţinerea situaţiilor de ieşire* presupune realizarea succesivă a operaţiilor:
- constituirea fişierelor tampon (de interogare) cu datele necesare obţinerii situaţiei;
- ordonarea fişierului tampon;
- generarea situaţiei (definirea structurii sale intr-un program sau cu generatorul de rapoarte);
- obţinerea efectivă a situaţiei.

4.6. IMPLEMENTAREA SISTEMELOR INFORMATICE

Implementarea finalizează activitatea de realizare a sistemelor informatice. În cadrul ei se testează, se asamblează, se verifică şi se asimilează de către beneficiar toate soluţiile stabilite în etapele anterioare şi se validează rezultatele obţinute.

Obiective ale etapei de implementare sunt: experimentarea sistemului proiectat; finisarea noului sistem; punerea în funcţiune; recepţia sistemului informatic.

Principalele grupe de activităţi ce trebuie realizate în etapa de implementare sunt:
- asigurarea condiţiilor de punere în funcţiune;
- funcţionarea experimentală şi punerea în funcţiune a sistemului proiectat;
- definitivarea documentaţiei sistemului informatic;
- recepţionarea şi omologarea noului sistem.

4.6.1. Asigurarea condiţiilor de punere în funcţiune

Implementarea sistemului informatic proiectat depinde în mod hotărâtor de felul în care beneficiarul asigură condiţiile de punere în funcţiune. Aceasta presupune în principal următoarele activităţi: difuzarea instrucţiunilor de executare a procedurilor manuale şi automate; instruirea personalului utilizator; asigurarea condiţiilor organizatorice necesare; asigurarea resurselor hard; asigurarea fondului informaţional.

a) Difuzarea instrucţiunilor de executare a procedurilor manuale şi automate. Aceste instrucţiuni se pot grupa în instrucţiuni pentru beneficiar (personalul de specialitate - economiştii din compartimentele funcţionale în sistemele economice - care va beneficia de rezultatele implementării) şi instrucţiuni pentru unitatea (colectivul) de informatică ce va asigura exploatarea noului sistem.

Instrucţiunile pentru beneficiar cuprind: prezentarea succintă a aplicaţiei şi a fluxurilor informaţionale şi decizionale; prezentarea noilor documente proiectate şi a instrucţiunilor de completare, verificare, avizare, pregătire pentru - prelucrare, şi dacă este cazul, şi a procedurilor de preluare pe suporţii tehnici de date; prezentarea situaţiilor de ieşire, a modului de folosire şi interpretare a acestora. Instrucţiunile pentru unitatea de informatică se referă la: modalităţile de prezentare şi recepţie a documentelor primare şi a situaţiilor cu rezultatele finale; modalităţi de pregătire şi verificare a purtătorilor tehnici de date precum şi de corectare a erorilor aferente; modalităţi de operare pe parcursul fluxului de prelucrare şi de intervenţie în cazul unor incidente ce pot să apară pe parcursul executării programelor.

b) Instruirea personalului utilizator grupează o serie de activităţi precum: iniţierea în utilizarea tehnologiilor informatice (sau actualizarea cunoştinţelor informatice); însuşirea corectă de către fiecare persoană implicată în funcţionarea noului sistem a sarcinilor ce îi revin, atât pe parcursul etapei de implementare, cât şi în exploatarea curentă; pregătirea psihologică a personalului unităţii beneficiare pentru recepţionarea corectă a efectelor pe care le va aduce cu sine sistemul informatic.

c) Asigurarea condiţiilor organizatorice necesare se referă în primul rând la transpunerea în practică a modificărilor organizatorice preconizate în etapa de proiectare generală, la lansarea efectivă a documentelor proiectate şi la instituirea noilor fluxuri informaţionale. De asemenea,

se va constitui nucleul de informaticieni ce va exploata noul sistem şi se vor planifica riguros toate activităţile specifice etapei de implementare.

d) Asigurarea resurselor hard se poate efectua în trei moduri:

- prin achiziţionarea tehnicii de calcul necesare, atunci când capacitatea de prelucrare poate fi acoperită cu lucrări;
- prin perfectarea accesului la o unitate prestatoare de servicii informatice pentru un anumit număr de ore-calculator, atunci când volumul lucrărilor este mic;
- printr-o soluţie mixtă, care presupune rezolvarea unor probleme, de un volum mai mic, în cadrul unităţii, iar pentru problemele mai complexe să se apeleze la capacitatea unei unităţi de informatică cu o dotare corespunzătoare, lucrându-se eventual în regim de teleprelucrare.

De o deosebită importanţă sunt şi: asigurarea softului de bază necesar; asigurarea materialelor consumabile; asigurarea unui spaţiu corespunzător desfăşurării activităţii colectivului de informatică, care să permită realizarea unor lucrări de calitate, dar şi păstrarea integrităţii şi confidenţialităţii fondului de date manipulat.

e) Asigurarea fondului informaţional. Presupune pregătirea datelor reale şi încărcarea fişierelor sau bazelor de date în vederea testării şi punerii în funcţiune a noului sistem, pregătire ce se face prin: constituirea fişierelor sau bazelor de date, prin culegerea fondului informaţional necesar şi stocarea acestuia pe purtători tehnici de date (pentru aceasta vor fi necesare şi o serie de activităţi pentru colectarea, ordonarea şi codificarea datelor); preluarea parţială sau integrală a datelor dintr-o serie de fişiere deja existente, cu ajutorul unor programe de conversie, dacă efortul depus corelat cu timpul necesar şi cu efectele obţinute justifică această operaţie.

4.6.2. Funcţionarea experimentală şi punerea în funcţiune a sistemului proiectat

Dacă în unitatea beneficiară există deja un sistem informatic sau dacă implementarea noului sistem se eşalonează mult în timp pe componente funcţionale, mai întâi se execută procedurile de conversie (pentru fişiere, programe, proceduri) din vechiul în noul sistem.

Responsabilitatea pentru punerea în funcţiune a unui sistem informatic revine: conducerii unităţii beneficiare, proiectantului sistemului informatic şi colectivului de informatică care preia sarcina prelucrării datelor în noul sistem.

Punerea în funcţiune a unui sistem informatic se face în funcţie de condiţiile concrete din sistemul economic. Din acest punct de vedere se pot întâlni mai multe situaţii:

- situaţia punerii în funcţiune a unui sistem informatic în cazul unui sistem informaţional nou, odată cu înfiinţarea unei noi societăţi comerciale sau a unei activităţi noi etc.
- situaţia punerii în funcţiune a unui sistem informatic pentru perfecţionarea unui sistem informaţional existent;
- situaţii de lansări în exploatare în funcţie de modul de asigurare atehnicii de calcul necesare (proprie; închiriată; proprie şi închiriată);
- situaţii de punere în funcţiune diferenţiate de complexitatea sistemului informatic, de sfera lui de cuprindere, de particularităţile agentului economi şi de dispersarea lui în teritoriu (sunt evidente diferenţierile ce există în punerea în funcţiune a unui sistem informatic la o societate comercială concentrată teritorial, faţă de una dispersată teritorial (unde organizarea fluxurilor informaţionale ridică probleme).

Se pot folosi mai multe strategii de implementare a unui sistem informatic, în funcţie de numărul şi ordinea subsistemelor care se testează şi de comparările ce se fac cu vechiul sistem.

În funcţie de ordinea de testare a subsistemelor componente implementarea poate fi:

- simultană, când toate subsistemele se testează odată, fără acordarea de priorităţi;
- în serie, care presupune testarea pe bază de prioritate a subsistemelor, de obicei testându-se mai întâi subsistemele cu frecvenţa cea mai mare.

În funcţie de compararea noului sistem cu vechiul sistem implementarea poate fi realizată în mai multe variante, dintre care enumerăm:

- implementarea directă cu date curente, care presupune renunţarea la vechiul sistem în vederea reducerii termenului şi a minimizării cheltuielilor de implementare;
- implementarea paralelă, care se face cu date curente sau anterioare, dar o perioadă noul sistem funcţionează în paralel cu cel vechi;
- implementarea pilotată, care presupune lansarea numai a anumitor subsisteme (de exemplu a celor cu frecvenţă maximă) folosind date din perioadele anterioare şi date curente şi efectuând comparări cu vechiul sistem.

De o deosebită importanţă este alegerea momentului punerii în funcţiune a unui sistem informatic care trebuie să corespundă cu începutul unei perioade semnificative din activitatea organismului economic în care acesta se integrează: începutul unui an calendaristic, introducerea unei noi forme de organizare, lansarea în producţie a unei noi tehnologii, începutul unei noi activităţi etc.

4.6.3. Documentaţia finală a sistemelor informatice

În timpul implementării unui sistem informatic pot să apară modificări (structurale şi funcţionale) care trebuie operate şi în documentaţia elaborată pentru a evita dificultăţile în exploatarea curentă şi întreţinerea ulterioară. Documentaţia finală a unui sistem informatic se concretizează în întocmirea următoarelor lucrări: manualul de prezentare; manualul de utilizare; manualul de exploatare (operare).

Manualul de prezentare. Cuprinde concepţia generală a sistemului şi se adresează conducerii unităţii beneficiare. Un manual de prezentare conţine în principal următoarele:

- elemente de identificare: pagina de titlu şi semnături; cuprinsul; baza elaborării; introducerea etc;
- obiectivele, performanţele şi limitele noului sistem;
- categorii de probleme abordate şi domeniul de aplicare;
- schema funcţională a sistemului informatic;
- locul sistemului în ansamblul sistemului informaţional al agentului economic (diagrama de relaţii);
- prezentarea ieşirilor: tipuri de ieşiri şi modul de obţinere; lista ieşirilor; exemple de ieşiri semnificative;
- prezentarea intrărilor: tipul de intrări şi moduri de completare; lista intrărilor;
- structura datelor: soluţia adoptată pentru organizarea şi gestiunea datelor; schema de structură generală a bazei de date şi a legăturilor între fişiere;
- schema generală a fluxurilor informaţionale în noul sistem;
- resurse necesare exploatării sistemului informatic: configuraţii de calcul, resurse umane, resurse financiare, alte resurse materiale auxiliare etc;
- condiţii de implementare;
- elemente de evaluare a sistemului informatic.

Manualul de utilizare. Este întocmit pentru fiecare subsistem în parte şi se adresează personalului implicat în utilizarea noului sistem la unitatea beneficiară. Instrucţiunile de utilizare se elaborează pe categorii de utilizatori (compartimente, persoane), iar gruparea pe aceste categorii se face de realizatorii sistemului informatic împreună cu conducerea unităţii beneficiare. În principiu, un asemenea manual cuprinde:

- elemente de identificare: pagina de titlu şi semnături; cuprinsul; baza elaborării; introducerea etc;
- proceduri manuale de codificare a datelor de intrare: structura codurilor pentru fiecare mulţime de date; graficul de implementare a codificării, modul de întreţinere şi responsabilităţi; liste de coduri şi cifre de control; legătura cu alte sisteme de codificare, la nivel naţional şi internaţional;
- proceduri manuale de colectare şi transmitere a datelor de intrare: prezentarea documentelor de intrare, a instrucţiunilor de completare, aprobare şi verificare (se

va anexa şi câte un exemplar din fiecare document completat cu date); instrucţiuni de stocare, arhivare, termene de păstrare, difuzare, distrugere; instrucţiuni de exploatare a procedurilor destinate preluării în sistemul de calcul a datelor de pe documentele de intrare de către utilizator; proceduri de control şi corecţie a datelor de intrare;

- proceduri de utilizare şi interpretare a ieşirilor: lista ieşirilor şi prezentarea unui exemplu din fiecare situaţie de ieşire; instrucţiuni de interpretare şi verificare; fluxuri de circulaţie a rapoartelor de ieşire, responsabilităţi, periodicitate, termene; modul de difuzare, arhivare, perioada de păstrare;
- proceduri speciale de conversie (valabile numai în etapa de implementare): lista procedurilor manuale şi automate de conversie; descrierea fiecărei proceduri.

Manualul de exploatare (operare). Cuprinde informaţii cu privire la exploatarea efectivă a sistemului proiectat prin intermediul sistemului de calcul şi se adresează personalului din unitatea de informatică. Are în componenţă următoarele piese:

- elemente de identificare: pagina de titlu şi semnături; cuprinsul; baza elaborării; introducerea etc;
- lista procedurilor şi graficul de exploatare a acestora;
- descrierea procedurilor de preluare, corectare, validare, stocare şi transmitere a datelor de intrare;
- descrierea tuturor procedurilor automate;
- descrierea procedurilor de operare la calculator: operaţii pregătitoare pentru efectuarea lucrării; moduri de intervenţie în caz de incident; lista mesajelor apărute pe parcursul operării şi modul de acţiune în fiecare caz; timpul de execuţie;
- proceduri de control şi transmitere a ieşirilor: controlul formal şi de fond al situaţiilor de ieşire; modul de corectare a erorilor şi de reluare a execuţiei; termene de predare la utilizator, evidenţa şi responsabilităţile pentru aceste predări.

4.7. EVALUAREA SISTEMELOR INFORMATICE

Teoria şi practica economică propun, în mod tradiţional, aprecierea oportunităţii şi utilităţii sistemelor informatice prin prisma eficienţei economice.

Analiza preliminară, în faza de stabilire a necesităţii realizării unui nou sistem, a resurselor ce vor trebui angajate, corelată cu cea a efectelor previzibile, precum şi urmărirea modului în care se cheltuiesc resursele, în concordanţă cu obiectivele realizate, este fără îndoială, ca în cazul oricărei investiţii, un lucru necesar dar nu suficient.

Având în vedere aceste considerente, trebuie să acceptăm ideea că evaluarea unui sistem informatic economic trebuie făcută prin prisma eficienţei economice, dar mai ales prin prisma modului în care sistemul informatic îşi realizează obiectivele, prin prisma calităţii.

4.7.1. Eficienţa economică a sistemelor informatice

Eficienţa unui sistem informatic poate fi calculată prin compararea efectelor cu valoarea resurselor alocate pentru construirea şi funcţionarea sa, dar, până în prezent, nu s-a ajuns la un punct de vedere comun cu privire la metodele de cuantificare.

Determinarea resurselor consumate pentru proiectarea, realizarea şi implementarea unui sistem informatic se poate face prin cumularea cheltuielilor în funcţie de etapele de proiectare parcurse şi categoriile de cheltuieli implicate.

În general, cheltuielile efectuate cu un sistem informatic pot fi grupate în două categorii**:**

a) cheltuieli de realizare, în care sunt cuprinse cheltuielile cu echipamentul, costul proiectului de realizare a sistemului informatic, costul amenajărilor auxiliare, cheltuieli cu pregătirea personalului etc.

b) cheltuieli de întreţinere – exploatare, în care intră costul utilizării calculatorului dacă acesta este închiriat, salariile personalului care va exploata sistemul informatic, costul energiei electrice consumate, al hârtiei de imprimantă, discurilor flexibile, costul pieselor de schimb, costul abonamentului de întreţinerea periodică etc.

Dacă din punct de vedere al elementelor de cost problemele sunt relativ clare, cuantificarea efectelor economice este mai dificilă. Efectele economice obţinute prin introducerea unui sistem informatic pot fi grupate în două categorii şi anume:

a) efecte directe, din cadrul sistemului informaţional cum ar fi cele rezultate din reducerea de personal, economia de formulare, rechizite etc.

b) efecte indirecte, din afara sistemului informaţional, şi anume cele rezultate din rezolvarea problemelor pe ansamblul unităţii economice.

Acest mod de abordare a efectelor economice a dus chiar la separarea eficienţei economice a sistemelor informatice în eficienţă economică directă şi eficienţă indirectă.

Teoria şi practica economică recomandă o serie de indicatori folosiţi în evaluarea eficienţei unui sistem informatic atât în faza de proiectare cât şi în faza de analiză. Aceşti indicatori pot fi grupaţi în două categorii şi anume:

a) Indicatori ai efectelor economice ce se concretizează în rezultate directe şi indirecte apărute în activitatea curentă a unităţii beneficiare (ca de exemplu: sporul de producţie; sporul de profit; economia de personal; reducerea cheltuielilor etc. - toţi calculaţi având în vedere nivelurile înregistrate înainte şi după introducerea sistemului informatic);

b) Indicatorii sintetici, care cuantifică eficienţa economică obţinută prin introducerea sistemului informatic ca o investiţie a unităţii economice beneficiare. Principalii indicatori în acest sens sunt:

1. Coeficientul eficienţei economice, care se poate calcula pentru fiecare subsistem informatic (k_{efj}) sau pentru sistemul informatic în ansamblu, se determină cu relaţia:

$$k_{efj} = \frac{E_{BJ}}{C_{RJ}}$$, în care:

E_{BJ} - efectele obţinute ca urmare a funcţionării sistemului (subsistemului J);

C_{RJ} - cheltuieli de realizare a sistemului (subsistemului J).

2. Termenul de recuperare a cheltuielilor totale cu realizarea unui subsistem informatic (T_{rj}) sau a sistemului informatic în ansamblu, care se exprimă în ani şi se determină astfel:

$$T_{rj} = \frac{1}{k_{efj}} = \frac{C_{RJ}}{E_{BJ}} \text{ (ani)}$$

3. Coeficientul eficienţei comparate (k_{ecomp}), calculat astfel:

$$k_{ecomp} = \frac{E_{B2} - E_{B1}}{C_{R2} - C_{R1}}$$, în care:

E_{B2}, E_{B1} - efectele obţinute în varianta de sistem informatic care presupune investiţie suplimentară, respectiv în varianta cu care se compară;

C_{R2}, C_{R1}- cheltuielile cu realizarea variantei de sistem informatic care necesită investiţii suplimentare, respectiv cheltuielile cu realizarea variantei cu care se compară.

4. Termenul de recuperare a investiţiei suplimentare (T_{rs}), exprimat în ani, care se determină cu relaţia:

$$T_{rs} = \frac{1}{k_{ecomp}} = \frac{C_{R2} - C_{R1}}{E_{B2} - E_{B1}} \text{ (ani)}$$

4.7.2. Managementul calităţii sistemelor informatice

În aprecierea calităţii unui sistem informatic trebuie pornit de la realitatea evidentă că sistemul este construit pentru client. Premisa de bază a calităţii sistemului informatic este aceea că ea trebuie abordată sistemic, începând cu fundamentarea deciziei de realizare a unui nou sistem, continuând cu organizarea şi conducerea realizării sistemului, cu implementarea acestuia,

pentru ca în final să putem aprecia calitativ funcţionarea efectivă a sistemului informatic.

Calitatea unui sistem informatic se poate privi din două puncte de vedere:

- pe de o parte, din punctul de vedere al utilizatorilor sistemului (aspectul funcţional);
- pe de altă parte, din punctul de vedere al celor ce au realizat sistemul (aspectul tehnic).

Din punct de vedere funcţional, ceea ce pare tehnic raţional şi corect poate să fie nefolositor utilizatorului. Aspectul tehnic porneşte de la premisa că, date fiind condiţiile dintr-o economie concurenţială, realizatorul unui sistem informatic trebuie să aleagă cele mai noi, cele mai performante soluţii. Absolutizarea unuia sau altuia dintre cele două aspecte poate să creeze serioase greutăţi în exploatarea sistemului informatic. Trebuie avut în vedere că sistemele informatice economice sunt create de oamenii pentru a fi utilizate în organizaţii de către oameni.

În literatura de specialitate se remarcă existenţa a două concepţii cu privire la calitate:

- *una orientată tehnic către producţie*, care cercetează existenţa unor abateri ale caracteristicilor produsului realizat faţă de nivelurile prevăzute în documentaţia tehnică întocmită înainte de începerea realizării produsului;

- *una orientată funcţional către utilizator*, care consideră calitatea ca fiind *aptitudinea unui produs sau a unui serviciu de a satisface nevoile utilizatorului*.

Problema calităţii materialelor (echipamentelor) utilizate este o problemă clasică de management al calităţii industriale, în timp ce problema elementelor logice este specifică domeniului. Norma ISO 9126 a enunţat şase caracteristici care permit descrierea calităţii unui sistem informatic:

- *Capacitatea funcţională* care cercetează dacă funcţiile pe care sistemul le realizează corespund nevoilor explicite sau implicite. Ea este descompusă în cerinţele: operabilitate, conformitate cu regulamentele, respect faţă de norme etc.

- *Facilitatea în utilizare* este definită în raport cu ansamblul utilizatorilor potenţiali şi se referă la efortul depus pentru a utiliza sistemul. Poate fi descompusă în facilităţi de comprehensiune, de iniţiere, de exploatare etc.

- *Fiabilitatea* este capacitatea sistemului de a-şi menţine, în condiţiile specifice utilizatorului, parametrii de performanţe pe toată perioada de exploatare stabilită şi se descompune în: maturitate, toleranţă la erori, posibilităţi de recuperare etc.

- *Randamentul (eficienţa* cercetează raportul între serviciile oferite de sistem (efectele sistemului) şi resursele utilizate într-un cadru dat.

- *Mentenabilitatea* care se referă la eforturile necesare pentru a menţine şi dezvolta sistemul, respectiv la posibilităţile de a aduce corecţii sau ameliorări funcţionale ulterioare, de a efectua schimbări organizaţionale sau de documentaţie etc. Este descompusă în: facilităţi de analiză, modularitate, stabilitate, facilitaţi de testare etc.

- *Portabilitatea* definită ca fiind capacitatea sistemului informatic de a fi transferat pe un mediu tehnologic diferit din punct de vedere fizic sau logic. Această caracteristică este descompusă în facilităţi de instalare şi facilităţi de adaptare.

Pentru perfecţionarea sistemelor informatice, pe lângă aprecierile făcute de specialişti în urma unei analize temeinice este nevoie şi de un sistem de indicatori care să permită estimarea calităţii. Pentru aceasta, se pot avea în vedere următorii indicatori:

- numărul de erori detectate în faza de implementare;
- numărul de erori detectate în exploatare;
- media gravităţii erorilor detectate
- intervalul mediu de timp necesar pentru remedierea unei erori:
- timpul mediu de acces la informaţii
- continuitatea în funcţionare
- numărul informaţiilor eronate rezultate din sistem,
- numărul cererilor clienţilor pentru care sistemul nu a fost capabil să dea un răspuns,
- timpul mediu necesar utilizatorilor pentru a învăţa să lucreze cu sistemul,
- numărul comenzilor pe care utilizatorul trebuie să le reţină etc.

CAP.5. PROGRAMARE ÎN INTERNET

5.1. INTERNET ŞI WEB. ARHITECTURI DE SISTEME DISTRIBUITE

5.1.1. Noțiuni generale

Cea mai răspândită reţea publică este Internet-ul. Internet-ul poate fi descris ca un *sistem deschis,* adica un sistem a carei arhitectura nu este secreta (producatorii sai i-au facut publica structura suficient de detaliat astfel încat alti dezvoltatori sa-şi poata interfata la el propriile produse). Câteva exemple de sisteme deschise sunt: sistemul de operare UNIX şi utilitarele asociate, protocoalele retelei Internet. Putem considera ca sistemele deschise au inspirat de asemenea tehnologiile bazate pe Java şi XML. Exemple de sisteme proprietare sunt: sistemele Microsoft Windows şi Mac OS cu utilitarele aferente.

Internet-ul are o arhitectură multinivel, inspirată de modelul de referință ISO/OSI.

În 1990 Tim Berners Lee a creat World Wide Web (WWW, WEB sau W3). Acesta a propus:

- O metodă de a atasa nume simbolice calculatoarelor din Internet
- O metodă de reprezenta documentele cu legaturi simbolice între ele (HTML)
- Conceptele de *server WEB* pentru stocarea acestor documente şi *navigator* (engl.*browser*) pentru afisarea acestor documente.

Internet-ul este o rțtea de sub-rețele. Sub-rețelele comunică între ele printr-un nod special numit *gateway*.

Protocoale ale retelei Internet:

- Telnet – protocol ce permite accesul la distanta la un calculator conectat la Internet, cu conditia ca utilizatorul sa aiba drept de acces la calculatorul respectiv
- FTP şi TFTP – protocoale de transfer de fisiere între calculatoare
- SMTP – protocol pentru transferul postei electronice între calculatoare
- Kerberos – protocol pentru transferul confidential de date între calculatoare
- SNMP – protocol pentru monitorizarea şi administrarea retelelor de calculatoare
- DNS – protocolul care permite denumirea simbolica a calculatoarelor conectate la Internet
- NFS – colectie de protocoale care permite accesul transparent la fisierele şi directoarele dintr-o retea de calculatoare
- TCP – protocol orientat pe conexiune ce permite transferul sigur şi fiabil al datelor între procesele aplicatiilor ce ruleaza pe calculatoare conectate la Internet
- UDP – protocol neorientat pe conexiune cu funcţie oarecum similara cu TCP, doar ca nu garanteaza transferul sigur al datelor
- IP – protocol ce asigura transferul pachetelor între calculatoarele conectate la Internet
- ICMP – protocolul de transfer a informaţiilor de comanda şi eroare între componentele retelei
- ARP şi RARP – protocoale ce asigura corespondenta directa şi inversa între adresele hardware şi adresele Internet ale calculatoarelor conectate la Internet

Adresarea în Internet

Principala funcţie a Internet-ului este schimbul de date între calculatoare. Pentru aceasta, fiecare calculator din Internet are asignata o *adresa IP*.

Versiunea cea mai raspindita la ora actuala a protocolului IP este IPv4. În IPv4 adresa unui calculator este un sir de 32 de biti. Acest sir se reprezinta sub forma unui 4-uplu format din 4 octeti, separati prin cate un punct. Exemplu: 151.23.40.3. Exista şi versiunea IPv6 care foloseste 128 de biti. Multimea de adrese IP este impartita în 5 clase:

- *Clasa A*, 0 adresa retea (7) adresa calculator (24)
- *Clasa B*, 10 adresa retea (14) adresa calculator (16)
- *Clasa C*, 110 adresa retea (21) adresa calculator (8)
- *Clasa D*, 1110 adresa multicast (28)
- *Clasa E*

Adresa 127.0.0.1 se mai numeşte şi adresă de *loopback*. Datele trimise aici se întorc înapoi la sursa. Aceasta este utilă pentru testarea aplicatiilor de retea folosind un singur calculator.

Adresele IP sunt dificil de memorat în format numeric, de aceea s-a introdus o metoda de a le atasa nume simbolice prin sistemul de numire a domeniilor (engl.*domain name system* – DNS). DNS reprezintă o modalitate de a denumi simbolic calculatoarele dintr-o retea bazata pe TCP/IP (Internet) folosind o schema de denumire ierarhica. Fiecare sufix dintr-un nume se numeste *domeniu*. Determinarea adresei IP pornind de la nume se face prin interogarea unui *server de nume* (engl.*name server*). DNS reprezintă multimea tuturor acestor servere.

Clienţi şi servere

Un *server* este un calculator (program în executie) din cadrul unei retele care furnizeaza servicii altor calculatoare (programe în executie) din cadrul retelei. Un c*lient* este un calculator (program în executie) dintr-o retea care beneficiaza de serviciile unui calculator (sau program) server.

În Internet, comunicarea între un client (program) şi un server (program) se face folosind suita de protocoale TCP/IP. Aceasta se bazeaza pe notiunile de *port* şi *soclu* (engl.*socket*). Un *port* reprezintă un canal abstract prin care un calculator (program care ruleaza pe calculator) poate comunica cu exteriorul. Un port este identificat printr-un numar. Porturile care aparţin intervalului 0…1023 sunt rezervate pentru servicii speciale. Cateva dintre acestea sunt:

- Port 7: ECHO
- Port 21: FTP
- Port 23: Telnet
- Port 80: HTTP
- Port 25: SMTP
- Port 110 :POP3
- Port 150 :SQL-NET

Un s*oclu* este o pereche formata dintr-o adresa de IP şi un port. Un soclu abstractizeaza notiunea de canal de comunicatie într-o retea bazata pe TCP/IP, usurand astfel programarea.

Tipuri de servere

- Servere de fisiere. Furnizeaza fisiere la cererea clientului; spre exemplu un depozit de documente (engl.*document repository*).
- Servere de baze de date. Stocheaza colectii mari de date structurate sub forma unor baze de date; furnizeaza servicii de interogare a acestora folosind SQL.
- Servere de *groupware*. *Groupware* = un sistem care permite unui grup de participanti sa lucreze impreuna într-un mediu partajat.

- Servere WWW. Sunt servere de fisiere care conţin componentele unui site din WWW. Accesul la ele se face printr-un program client special numit *navigator*.
- Servere de posta electronica. Permit receptia, stoarea şi trimiterea de mesaje prin posta electronica.
- Servere de obiecte. Stocheaza obiecte şi permit programelor client sa trimita mesaje acestor obiecte.
- Servere de imprimare. Furnizeaza clientilor servicii de imprimare.
- Servere de aplicatii. Sunt servere dedicate uneia sau mai multor aplicatii particulare şi conţin programele dedicate aplicatiei respective.

5.1.2. World Wide Web

WWW este un *sistem hipermedia distribuit*. Se bazeaza pe un model de structurare a documentelor care foloseste trei concepte:

- ***Multimedia*** **–** se refera la integrarea mai multor tipuri de media în cadrul aceluiasi model de document: text, grafica, imagine, video, etc.
- ***Hiperdocument*** **–** se refera la crearea de legaturi între documente, folosind un mecanism propriu modelului de document.
- ***Documente distribuite*** **–** se refera la documente care conţin legaturi la documente stocate pe alte calculatoare din cadrul unei retele.

Se spune ca WWW foloseste un *model de documente hipermedia distribuite*. Termenul de hipermedia înglobeaza conceptele de multimedia şi hiperdocument.

Fiecare autor care creeaza o resursa informaţionala (engl.*informaţion resource*) în WWW o considera ca fiind un document separat. Însa putem considera ca, la nivel global, multimea tuturor resurselor informaţionale din WWW formeaza un unic document hipermedia distribuit. Din acest punct de vedere, termenul de resursa informaţionala este mai potrivit decat cel de document. Exista trei nivele de distribuire a resurselor informaţionale în WWW: acelasi fisier, fisiere separate pe acelasi calculator, calculatoare diferite.

Desi WWW a aparut în 1990, conceptele de baza au ramas aceleasi pana astazi: URL (denumirea unui document), HTTP (regasirea unui document) şi HTML (descrierea conţinutului unui document).

Arhitectura WWW este o arhitectura client/server tipica şi anume:

- Un server WWW are sarcina de a gestiona o multime de documente din cadrul WWW. Aceste documente se numesc şi *pagini WWW*.
- Un client generic de WWW este un program care emite cereri către un server WWW pentru accesarea paginilor WWW gestionate de acel server. Exemple de clienti sunt:
  - *Un navigator WWW care permite regasirea şi afisarea paginilor WWW în scopul vizualizarii conţinutului lor de către un agent uman.*
  - *Un program de tip softbot care localizeaza diverse pagini WWW în scopul crearii unui index. Indexul poate fi utilizat ulterior de un motor de cautare.*

Conceptele pe care se bazeaza tehnologia WWW sunt:

- Schema de denumire a resurselor (engl.*uniform resource locator*) URL
- Protocolul de transfer al documentelor (engl.*hypertext transfer protocol*) HTTP
- Limbajul de specificare a conţinutului paginilor WWW (engl.*hypertext markup language*) HTML

Pentru identificarea resurselor în WWW se foloseste un URL (engl.*Uniform Resource Locator*). Un URL este un identificator simbolic al resursei şi este compus din doua parti:

- Schema, care indica modalitatea folosita pentru denumirea resurselor. Spre exemplu, schema poate fi numele unui proocol: *ftp*, *http*, etc
- Partea specifica schemei, care indica cum se adreseaza resursa în cadrul schemei respective

Sintaxa URL este **schema ":" parte-specifica-schemei**. Partea specifica schemei are sintaxa "//" [*utilizator* [":" *parola*] "@"] *gazda* [":" *port*] "/" *cale*

- *utilizator* şi *parola* sunt optionale şi se aplica doar cu schemele care au sens (de exemplu *ftp*).
- *gazda* indica numele calculatorului pe care se afla resursa, sau adresa de IP a acestuia.
- *port* reprezinta numarul portului pe care se face conexiunea. Este optional, deoarece acest numar este predefinit pentru serviciile standard. Pentru HTTP portul predefinit este 80.
- *cale* reprezinta calea de acces la resursa în cadrul calculatorului specificat. În general este o cale fizica existenta în sistemul de fisiere de pe calculatorul gazda.

Exemple de aplicatii WWW:

- Furnizarea de adrese de email anonime pentru trimiterea de *spam*. Termenul de *spam* se refera la trimiterea de mesaje de email cu anunuturi unor receptori ce nu au solicitat primirea mesajelor respective. *Spam*-ul este detestat de toti utilizatorii de Internet.
- Site-uri de verificare automata a validitatii/actualitatii unor legaturi WWW.
- Site-uri de arhivare pentru stocarea sigura de fisiere.
- Site-uri care furnizeaza adrese de email gratuite. Sunt utile pentru utilizatorii mobili.
- Motoare de cautare (eng.*search engines*). Cautarea se face într-o baza de date uriasa care conţine informaţii despre documentele din WWW, organizate sub forma unui index. Specificarea cautarii se face printr-o interogare (engl.*query*). Baza de date este construita cu ajutorul unui program special numit *spider* (un client special de HTTP) care navigheaza automat pe WWW şi stocheaza în indexul local informaţiile relvante gasite. *Spamdexing* este o tehnica folosita de unele site-uri pentru a fi gasite ca relevante la cautarea unor cuvinte cheie raspandite.
- Grupuri de utilizatori interesati de un subiect comun – *newsgroup* şi *mailing list*.
- Aplicatii de comert electronic

5.1.3. Protocolul HTTP

HTTP este protocolul de comunicare între clientii şi serverele WWW. El specifica cererile care pot fi adresate de clienti şi raspunsurile care pot fi generate de servere. Specificatia conţine *structura* şi *formatul* (sintaxa) acestora.

Cererile şi raspunsurile HTTP se mai numesc şi *mesaje*. Un mesaj este un sir de caractere ce conţine: *linia de start* (engl.*start line*), zero sau mai multe *campuri antet* (engl.*message header field*) şi optional un *camp corp* (engl.*message body field*).

Antetele unui mesaj pot fi: *antete generale* (engl.*general header*) – se refera la mesaj, nu la entitatea transmisa; *antete de entitate* (engl.*entity header*) – se refera la entitatea transmisa sau referita; *antete raspuns* (engl.*response header*) – serverul comunica clientului informaţii care nu au fost incluse in linia de start.

Cereri – Linia de start se numeste *linie de cerere* (engl.*request line*) şi are structura:
metoda spatiu url_resursa spatiu versiune_http sfarsit_linie

Raspunsuri – Linia de start se numeste *linie de stare* (engl.*status line*).

Referitor la sintaxa formala a unei cereri HTTP, trebuie precizate urmatoarele:

- HTTP 1.0 este descris la nivel informaţiv în documentul RFC 1945.
- HTTP 1.1 este descris în propunerea de standard RFC 2068.
- O *cerere* HTTP are structura urmatoare ([...] specifica un element optional şi * specifica 0 sau mai multe repetitii ale unui element):

cerere =

linie-de-cerere
(*antet-general* | *antet-cerere* | *antet-entitate*)*
sfarsit-de-linie
[*corp-mesaj*]
linie-de-cerere = metoda spatiu uri spatiu versiune-http sfarsit-de-linie
metoda =OPTIONS | GET | HEAD | POST | PUT | DELETE |
TRACE | CONNECT | *metoda-extensie*
uri = * | *uri-absolut* | *cale-absoluta*

– Un *raspuns* HTTP are structura urmatoare:

raspuns =

linie-de-stare

(*antet-general* | *antet-raspuns* | *antet-entitate*)*

sfarsit-de-linie

[*corp-mesaj*]

linie-de-stare = versiune-http spatiu cod-stare spatiu fraza-motiv sfarsit-de-linie
cod-stare = cod-succes | *redirectare* |

eroare-client | *eroare-server* | ...

cod-succes = ok | *creat* | *acceptat* | ...
ok = 200
creat = 201
acceptat = 202
redirectare = mutat-permanent | *mutat-temporar* | ...
mutat-permanent = 301
mutat-temporar = 302
eroare-client = cerere-gresita | *neautorizat* | *plata-necesara* | *interzis* |
resursa-inexistenta | *metoda-nepermisa* | ...
cerere-gresita = 400
neautorizat = 401
plata-necesara = 402
interzis = 403
resursa-inexistenta = 404
metoda-nepermisa = 405
eroare-server = eroare-interna | *nu-este-implementata* | ...
eroare-interna = 500
nu-este-implementata = 501

Cookies

HTTP este un protocol *fara stare* (engl.*stateless*). Acest lucru înseamna ca HTTP nu dispune de un mecanism propriu care sa permita unui server WWW sa poata pastra informaţii despre starea utilizatorului. Termenul de *cookie* se refera la mecanismul prin care o aplicatie WWW pe partea de server poate stoca şi regasi informaţii pe partea de client. El permite adaugarea adaugarea unei stari a conexiunii stocata la client. Serverul specifica clientului ca vrea sa stocheze un *cookie* prin antetul Set-Cookie, în maniera urmatoare:

Set-Cookie: *Nume=Valoare*

Ulterior, clientul va include cookie-ul într-un antet al unei cereri sub forma:

Cookie: *Nume=Valoare*

Suplimentar, antetul Set-Cookie mai poate conţine urmatoarele informaţii:

- domain: specifica domeniul în care se aplica cookie-ul
- expires: specifica data de expirare a cookie-ului
- path: specifica URL-urile la care clientul va returna cookie-ul în cererea HTTP
- secure: specifica faptul ca cookie-ul va fi returnat de client numai dacă conexiunea este sigura.

Ex: Set-Cookie: Credit=111; secure; expires=Thursday, 07-Dec-2000, 10:00:00 GMT; domains=.comp-craiova.ro; path=/

Relativ la trimiterea de date către un server WWW, trebuie precizat ca :

- Paginile de WWW pot fi *statice* şi *dinamice*. *Paginile statice* sunt stocate explicit pe server şi returnate clientilor ca raspuns la cereri GET. *Paginile dinamice* sunt generate dinamic de server, nefiind stocate explicit. Este posibil ca aceste pagini sa fie configurate pe baza unor date trimise de la client către server. Astfel de date pot fi de exemplu criterii de cautare într-o baza de date, paginile generate dinamic fiind în acest caz rapoartele rezultate în urma cautarii.
- Exista doua metode de a transmite date de la client la server:
  - Folosind metoda GET, datele sunt atasate URL-ului sub forma unor perechi variabila-valoare. În acest caz URL-ul reprezinta un program aflat pe server şi datele sunt transmise acestui program în mediul sau de executie (engl.*environment*). La URL se adauga un sir de forma:
    ?nume$_1$=valoare$_1$&nume$_2$=valoare$_2$& ... &nume$_n$=valoare$_n$
    Un ' ' este codificat prin '+' şi un caracter special printr-un cod hexa precedat de '%'. Ex. ?cale=%2Fweb%2 semnifica valoarea /web/ pentru variabila *cale*.
  - Folosind metoda POST, datele sunt atasate în corpul cererii, spre deosebire de cazul anterior, unde sunt atasate URL-ului, în antet. Ele sunt transmise programului prin intrarea standard.
- Serverul preia datele de la programul invocat prin iesirea standard şi le trimite apoi clientului. Aceasta tehnica de a extinde funcţionalitatea unui server WWW pentru generarea dinamica de pagini se numeste Common Gateway Interface (CGI).

Middleware

Middleware este un term destul de vag ce se refera în general la toate nivelurile software intermediare care sprijina comunicatia dintre un client şi un server.

Un middleware furnizeaza un set standard de interfete pentru o colectie de resurse distribuite disparate, eterogene şi proprietare. Astfel dezvoltatorii isi vor interfata aplicatiile cu partea de middleware în loc de interfetele de nivel coborat ale resurselor proprietare. Un exemplu de middleware este software-ul care interfateaza un program navigator de sistemul WWW.

O tehnologie foarte raspandita este middleware-ul orientat pe mesaje – MOM (engl.*message-oriented middleware*). MOM gestioneaza tranzactiile dintre un client şi un server prin intermediul unor cozi care stocheaza mesajele transmise între clienti şi serveri.

Un exemplu de MOM este *WebSphere MQ* dezvoltat de IBM (fost *MQSeries*). *WebSphere MQ* gestioneaza transferul de mesaje între clienti şi serveri şi stie sa prelucreze patru tipuri de mesaje: datagrame – mesaje unidirectionale, mesaje de cerere, mesaje de raspuns şi mesaje de raportare. *WebSphere MQ* este un lider în domeniul platformelor middleware pentru integrarea aplicatiilor de e-business.

5.1.4. Arhitecturi multistrat (engl.*tiered architectures*)

O *aplicatie distribuita* este compusa dintr-o multime de programe care ruleaza pe mai multe calculatoare conectate în retea. O schita a acestor programe impreuna cu calculatoarele (engl.*hosts*) pe care ruleaza, responsabilitatile lor şi protocoalele prin care comunica se numeste *arhitectura distribuita*.

O clasificare a arhitecturilor distribuite se bazeaza pe conceptul de *strat* (engl.*tier*). Un strat poate fi un calculator (însa putem avea aplicatii distribuite virtual care ruleaza pe un acelasi calculator) sau o partitie logica de prelucrare din cadrul aplicatiei. De obicei un strat corespunde unui client sau server.

Avantaj: paradigmele de codificare sunt diferite de la strat la strat astfel ca straturi diferite cer îndemanari de programare diferite

Partitionarea logica a unei aplicatii distribuite trebuie sa aiba în vedere cel putin urmatoarele trei elemente:

- Logica de prezentare
- Logica problemei (engl.*business logic*)
- Logica datelor, responsabila cu persistenta datelor, controlul accesului concurent, corectitudinea tranzactiilor, etc

Avantaj: straturile permit separarea elementelor enumerate mai sus

Clasificarea arhitecturilor multistrat:

- 1 strat (engl.*one tier*)
  - Este simpla deoarece nu exista conectare în retea
  - Performante bune, deoarece nu exista comunicatii în retea
  - Sistemul este autoconţinut
  - Nu exista posibilitatea accesului de servicii la distanta
  - Arhitectura monolitica, deci potential de “cod spaghetti”
- 2 straturi (engl.*two tiers*)
  - Straturi: client şi server de WWW, arhitectura oarecum simpla
  - Separa logicii de prezentare de logica problemei
  - Potential mic pentru partajarea resurselor, o problema în comertul electronic
- 3 straturi (engl.*three tiers*)
  - Straturi: client, server de WWW, server de baze de date
  - Separa logica de prezentare, logica problemei şi logica datelor
  - Necesita expertiza în plus, maparea obiectual-relational este destul de dificila

- 4 straturi (engl.*four tiers*)
  - Straturi: client, server WWW, server de aplicatii, server de baze de date
  - Flexibilitate ridicata, practic poate realiza orice
  - Nivel de expertiza foarte înalt, curba de învatare mare, cost foarte ridicat, poate fi ineficienta datorita generalitatii
  - Partea dintre logica de prezentare (client) şi logica datelor (baza de date) se mai numeste şi strat intermediar (engl.*middle layer)*. El conţine printre altele obiectele problemei (engl.*business objects*), ce corespund entitatilor din domeniul problemei

Putem avea şi arhitecturi cu $n \geq 4$ straturi (engl.*n-tiers*). Cu cat numarul de straturi este mai mare, cu atat performantele pot sa scada, implementarea este mai dificila şi cere expertiza mai mare, complexitatea sistemului este mai mare, costul total al sistemului creste. În consecinta, stabilirea numarului de straturi trebuie facuta cu grija, în funcţie de cerintele reale ale aplicatiei. Cel mai adesea o arhitectura în 3 straturi este suficienta. Serverele de aplicatii sunt în general foarte scumpe şi curba de învatare este foarte lenta.

5.1.5. Paradigme de programare distribuita

1. Modelul transferului de mesaje:

- Se bazeaza pe ideea de *protocol*. Acesta poate fi gandit ca un limbaj ce întruchipeaza funcţiile cerute de un client şi pe care le furnizeaza un server.
- Protocoalele se impart în:
  - *Protocoale fixe*, al caror vocabular este fixat şi incapsulat în codul serverului şi clientului.
  - *Protocoale adaptive*, care se pot schimba la momentul executiei
- Un exemplu de protocol fix este cel pentru comunicarea cu un server de nume:
  - Partea de client: CAUTA *Nume,* STERGE *Nume,* ADAUGA *Nume, Resursa,* MODIFICA *Nume, Resursa*
  - Partea de server: RESURSA *Detalii,* STERGE OK, ADAUGA OK, MODIFICA OK, EROARE *Cod*
- O metoda de a implementa protocoalele adaptive este folosirea obiectelor serializabile. Acestea sunt obiecte care pot fi transferate în retea sub forma de date. Un astfel de obiect poate conţine funcţionalitatea unei noi comenzi din cadrul protocolului.
- În cadrul acestui model clientii şi serverii comunica prin transfer de mesaje de-a lungul unor canale de comunicatie. Corespunde oarecum cu structura retelei în care ruleaza clientii şi serverii. Scopul acestui model este de a abstractiza detaliile de nivel coborat şi de a face astfel programarea aplicatiilor mai usoara.
- Se preteaza în urmatoarele situatii:
  - Cerintele de comunicare sunt foarte simple
  - Sunt necesare performante foarte bune; acest model este cel mai eficient dintre modele discutate; plata pentru aceasta eficienta este cresterea complexitatii programarii
- Exemplu: HTTP, protocolul de comunicare cu un server de WWW
- Exista doua clase:
  - Transfer *sincron* de mesaje: entiatea *A* trimite un mesaj către entitatea *B*, în timp ce entitatea *B* prelucreaza mesajul, entitatea *A* se opreste şi asteapta un raspuns, după ce entitatea *B* raspunde entitatea *A* isi conţinua activitatea
  - Transfer *asincron* de mesaje: entiatea *A* trimite un mesaj către entitatea *B*, in timp ce entitatea *B* prelucreaza mesajul, entitatea *A* isi conţinua activitatea, cand entiatea B raspunde, entitatea *A* poate prelua mesajul.

2. Modelul obiectelor distribuite:

- Se numeste *obiect distribuit* un obiect rezident pe un calculator care poate fi invocat de obiecte rezidente pe alte calculatoare astfel încat, dpdv al programatorului, obiectele sa para a

fi situate pe un acelasi calculator (adica toate detaliile de comunicare sa fie ascunse programatorului).
- Tehnologia obiectelor distribuite presupune urmatoarele:
  - Interceptarea apelurilor către obiecte
  - Localizarea obiectelor
  - Comunicarea mesajelor şi parametrilor către aceste obiecte
  - Optional, returnarea eventualelor rezultate şi transmiterea lor obiectului apelant
- Avantaj: obiectele distribuite corespund 100% modelului orientat pe obiect şi din acest motiv trecerea de la proiectare la implementare este usoara
- Dezavantaj: performantele tehnologiilor de obiecte distribuite sunt inferioare tehnologiilor de transfer de mesaje
- Exemple
  - CORBA, propus de OMG
  - Apelul metodelor la distanta (engl.*Remote Method Invocation*) Java RMI
  - DCOM, propus de Microsoft

3. Modelul evenimentelor:

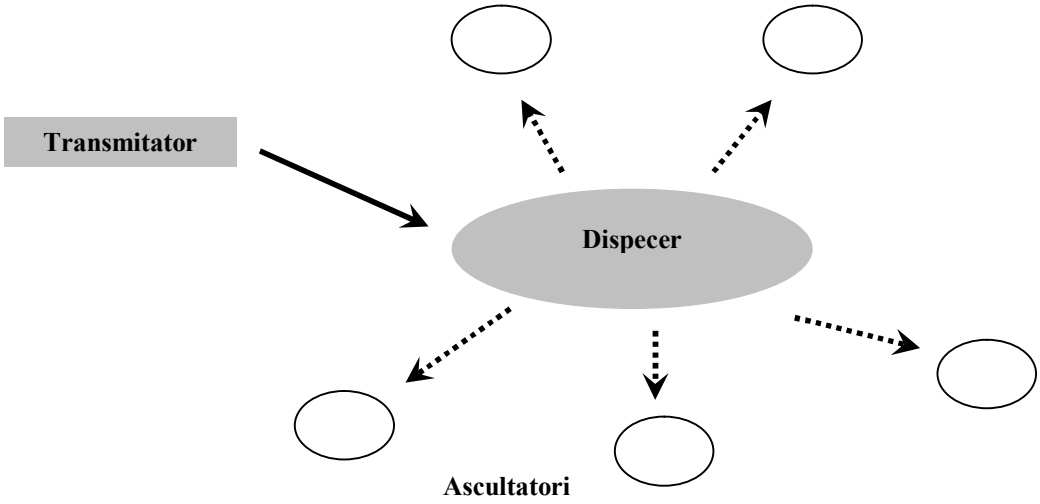
- Acest model presupune asocierea unor portiuni de cod cu anumite evenimente. Codul va fi executat automat la declansarea evenimentelor respective.
- Acest model de programare este foarte raspandit în programarea interfetelor grafice. Dezvoltarea unei astfel de interfete presupune urmatoarele:
  - Controale grafice, spre exemplu butoane, sunt plasate într-un container de tip fereastra principala, numita şi cadru (engl.*frame*)
  - Fereastra cadru implementeaza o interfata prin intermediul careia va fi invocata la declansarea unor evenimente. Codul de tratare a evenimentelor va fi amplasat în metodele ce implementeaza aceasta interfata.
  - Fereastra cadru se înregistreaza la controalele grafice ca obiect ascultator (engl.*listener*), ceea ce semnifica faptul ca este interesata de a fi informata la declansarea anumitor evenimente.
  - La declansarea unui eveniment, controlul grafic informeaza toate obiectele ascultator înregistrate la el.

Magistrala de obiecte:

- Modelul evenimentelor este folosit şi în arhitecturile de tip *magistrala de obiecte* (engl.*object bus*)
- Într-o magistrala de obiecte, serverele imping datele către clienti, din acest motiv folosindu-se şi terminologia *push tehnology*. Spre deosebire de aceasta, în obiectele distribuite clientii extrag datele de la serveri, folosindu-se terminologia *pull technology*.
- În magistrala de obiecte, transmitatorul este server şi ascultatorii sunt clienti.
- Arhitectura magistrala de obiecte este utila în aplicatiile în care evenimentele se declanseaza în timp real iar multimea de ascultatori se poate schimba dinamic. Exemple:
  - Furnizarea de date despre piata de capital institutiilor financiare interesate
  - Teleconferinte sau aplicatii conversationale (engl.*chat room*), unde mesajele între participanti se transmit în timp real
  - Aplicatii multimedia distribuite de tip video la cerere (engl.*video on demand*), unde un mare volum de date trebuie transmis în timp real abonatilor
- Exista doua mari clase de arhitecturi de tip magistrala de obiecte:
  - Arhitectura *butuc şi spite* (engl.*hub and spoke*)
  - *Magistrala cu multitransmisie* (engl.*multicast bus architecture*)

Arhitectura butuc şi spite:

- Transmitatorul trimite date către dispecer (butuc), iar acesta le distribuie ascultatorilor înregistrati. Poate exista cate un dispecer pe canal sau un singur dispecer pentru toate canalele.
- Avantaje: usor de implementat folosind socket-ui sau RMI, contabilizarea poate fi centralizata la dispecer
- Dezavantaje: se genereaza un volum mare de trafic comparativ cu magistralele cu multitransimise, fiabilitate scazuta deoarece totul depinde de dispecer



Magistrala cu multitransmisie:

- Tehnica multitransmisiei permite transferul unui singur mesaj de la transmitator către mai multi receptori.
- Mesajul este transmis pe o magistrala, de unde este preluat de toti ascultatorii interesati; ascultatorii sunt activati printr-un eveniment care ii informeaza ca mesajul este disponibil pe magistrala.
- Un exemplu este *iBus* de la *SoftWired Ltd* care poate fi gandita ca un fel de implementare software a protocolului Ethernet. Astfel mesajele vor fi preluate numai de destinatarii interesati; dacă mesajul nu este necesar este pur şi simplu ignorat şi lasat sa treaca urmatorului destinatar conectat la magistrala.

4.Modelul tuplelor:

- A fost folosit în limbajul de programare de nivel înalt Linda, dezvoltat de cercetatorii americani Nicolas Carriero and David Gelerntner în 1980. Acesta a fost raspandit doar în mediul academic. Firma Sun s-a inspirat din el in 1990 pentru a dezvolta tehnologia JavaSpaces ca parte a proiectului JINI. JINI este o arhitectura deschisa pentru interconectarea în retea a unei multimi de servicii implementate soft sau hard şi care poate fi reconfigurabila dinamic, iar JavaSpaces este modelul de programare distribuita folosit în JINI.
- In JavaSpaces procesele comunica prin intermediul unor zone partajate numite *spatii* (engl.*spaces*). Clientii pot accesa aceste spatii prin 3 operatii:
  - *write*, pentru adaugarea unui nou obiect la un spatiu
  - *take*, pentru citirea unui obiect şi eliminarea sa dintr-un spatiu
  - *read*, pentru crearea unei copii a unui obiect dintr-un spatiu
- Spatiile sunt containere de obiecte cu urmatoarele proprietati:
  - Sunt partajate şi pot fi accesate concurent
  - Sunt persistente

- Sunt asociative adica obiectele sunt localizate prin *regasire asociativa* Tranzactiile pe aceste spatii sunt sigure
- Spatiile permit schimbul de conţinut executabil

5.2. PROGRAMARE PE PARTE DE CLIENT

Termenul de *programare pe partea de client* (engl.*client side programming*) se refera la executarea de programe la client în scopul cresterii gradului de interactivitate al paginilor WWW. Programele destinate a fi executate la client se pot transmite de la server către client fie în *format sursa*, fie în *format obiect*.

Termenul de *programare pe partea de server* (engl.*server-side programming*) se refera la executarea de programe la server în scopul de a genera date şi/sau rapoarte care sunt trimise clientului sau mai general, pentru extinderea funcţionalitatii unui server de WWW. Datele şi rapoartele sunt trimise clientului de obicei în format HTML.

5.2.1. Limbajul HTML

Limbajul HTML reprezintă *Lingua franca* pentru publicarea de hipertext în pagini WWW. HTML este inspirat din SGML - un *limbaj de meta-marcare* (engl.*meta-markup*). În esenta, acesta permite definirea de noi *tipuri de documente* printr-o *definitie de tip de document* – DTD. Unui tip ii corespunde un *limbaj de marcare* specific (ex. HTML), ce descrie conţinutul şi structura unui document. Un document scris într-un limbaj de marcare conţine:

- *Date* = conţinutul propriu-zis al documentului.
- *Marcaje* = informaţii referitoare la structura documentului. Se numesc şi *metadate*.

Versiunea 4.0 defineste 3 tipuri de documente HTML:

- Documente *tranzitionale*: acest tip se va folosi doar pentru verificarea documentelor HTML, în nici un caz pentru generarea de noi documente. Astfel, elemente ca BASEFONT, FONT, CENTER, S, STRIKE sau U sunt doar parte a documentelor tranzitionale, folosite pentru formatare.
- Documente *stricte*: acest tip nu include elementele prezente în documentele tranzitionale pentru compatibilitate cu versiunile anterioare de HTML.
- Documente tip *colectii de pagini*

Tipul de document se defineste printr-o *definitie de tip* (concept preluat din SGML). Definitia de tip se specifica optional la începutul unui document HTML astfel:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
http://www.w3c.org/TR/REC-html40/strict.dtd

Structura unui document HTML

Un document HTML conţine date şi marcaje (engl.tag). Marcajele conţin informaţia de formatare, iar datele reprezinta conţinutul efectiv al documentului. Marcajele sunt interpretate de către programul navigator, ele determinand modul în care vor fi afisate datele. Marcajele pot fi: marcaje de început (engl.start tag) şi marcaje de sfarsit (engl.end tag).

Un marcaj de început poate conţine optional atribute cu valori şi se scrie astfel:

<*marcaj atribut* = ”*valoare*” *atribut* = ”*valoare*” …>

Ex: <img src= ”logo.gif” alt=”home page” >

Un marcaj de sfarsit se scrie astfel:

</*marcaj*>

Ex: </img>

Teoretic, marcajele trebuie închise corect, exact ca parantezele dintr-o expresie matematica. În practica însa, deseori, marcajele de sfarsit se omit, ambiguitatile fiind rezolvate prin metode ad-hoc de programul navigator. În principiu, un document HTML conţine un *antet* (engl.*header*) şi un *corp* (engl.*body*) între marcajele <html> şi </html>. html se numeste şi element radacina al documentului.

Elementele din corpul unui document HTML se clasifica în:

- Elemente de *nivel bloc* (engl.*block-level*). Ex: P, H, UL, OL, PRE, TABLE, FORM
- Elemente *incluse* (engl.*inline*). Ex: I, B, U, EM, STRONG, A, IMG

Diferenta între ele se bazeaza pe trei coordonate: conţinut, formatare şi directionalitate. Relativ la conţinut, trebuie menţionate următoarele aspecte:

- Un element de nivel bloc poate conţine atat elemente incluse cat şi elemente de nivel bloc.
- Un element inclus poate conţine numai elemente incluse şi date.
- Aceasta distinctie sugereaza ca elementele de nivel bloc pot crea structuri mai bogate decat elementele incluse.

În privinţa formatării, trebuie să se ţină cont că:

- Elementele de nivel bloc sunt formatate întotdeauna începand cu o linie noua, în timp ce elementele incluse nu.
- Elementele incluse pot fi despartite pe mai multe linii (la fel cum textul unui paragraf se desparte în linii), în timp ce elementele de nivel bloc nu.

Directionalitatea se refera la directia în care este scris textul şi se respectă regula următoare:

Elementele de nivel bloc mostenesc directionalitatea de la elementul cuprinzator, în timp ce elementele incluse nu.

Un exemplu de document HTML este urmatorul :

```
<HTML>
<HEAD>
<TITLE> Un exemplu simplu de pagina WWW </TITLE>
</HEAD>
<BODY>
<H1> Amelia Badica despre HTML </H1>
Un document HTML conţine:
<UL>
   <LI> marcaje - marcajele pot fi:</LI>
   <UL>
      <LI>marcaje de inceput</LI>
      <LI>marcaje de sfarsit</LI>
   </UL>
   <LI> date </LI>
</UL>
```

şi este compus din:
<UL>
  <LI> un antet </LI>
  <LI> un corp </LI>
</UL>
</BODY>
</HTML>

Formatarea textului în HTML

Textul HTML poate fi *simplu* sau *structurat*.

- *Textul simplu* este format din *paragrafe* (p), separate de linii libere (br) sau *titluri* (engl.*headings*, h1, h2,…, h6). Textul poate avea stiluri (ex. bold b, italic i, etc), fonturi (font, basefont)
- *Textul structurat* cuprinde *liste* şi *tabele*.

- *Listele* sunt: *liste neordonate* (ul, li) cu elementele indicate cu simboluri nenumerice (engl.*bullet*), *liste ordonate* (ol, li) cu elementele numerotate, şi *liste de definitii* (dl, dd) cu simbolurile elementelor definite de utilizator.
- *Tabelele* (table) permit structurarea textului în linii şi coloane. În principiu un tabel conţine unul sau mai multe *corpuri de tabel* (engl.*table body*, tbody). Un corp de tabel conţine una sau mai multe *linii* (engl.*row*, tr). O linie este formata din celule. Acestea pot fi *celule antet* (engl.*header cell*, th) sau *celule de date* (engl.*data cell,* td).

Pentru tabele exista şi alte elemente şi atribute optionale care specifica titlul tabelului (caption), alinierea celulelor (align), stilul conturului (border), formatarea textului în celule, etc. Spre exemplu stilul (grosimea) conturului se specifica astfel: <table border=”1”>.

Legaturi HTML

Legăturile HTML se bazeaza pe folosirea de *ancore* (engl.*anchor*) cu ajutorul elementului a. Exista doua tipuri de ancore: *ancore sursa* şi *ancore destinatie*.

Ancorele sursa au forma

text

şi reprezinta originea unei legaturi. *url* reprezinta resursa destinatie a legaturii. *text* este textul afisat in zona sensibila a legaturii.

Ancorele destinatie au forma

şi reprezinta destinatia unei legaturi. Numele unei ancore destinatie poate fi specificat intr-o ancora sursa prin #*nume* pe post de *url*. Ancorele destinatie se pot specifica şi cu atributul *id* in cadrul elementului referit, in acest caz numai fiind nevoie sa se foloseasca elementul a.

Documente HTML multifereastra

HTML 4.0 permite definirea de *colectii de pagini* (engl.*frameset*) pentru integrarea unei multimi de documente intr-o aceeasi fereastra principala. Fiecare document va fi vizualizat intr-o portiune a ferestrei principale.

Pentru definirea structurii multifereastra se foloseste un document HTML special numit *tipar* (engl.*frameset document*) ce conţine marcajele speciale frameset, frame şi noframes (optional). Celelalte documente ale colectiei sunt documente HTML obisnuite (engl.*frame document*).

Documentul tipar specifica documentele care se incarca in fiecare fereastra. Dacă aceste

documente conţin legaturi spre alte documente, in cadrul acestor legaturi se poate specifica fereastra destinatie prin atributul target. Numele ferestrelor se specifica cu atributul name in marcajul frame.

Specificarea dimensiunii ferestrelor in documentul tipar se face cu atributele rows şi cols. Pentru fiecare fereastra se indica fie dimensiunea absoluta (numar), fie procentul din spatiul total (procent) fie dimensiunea relativa (numar şi *).

Se pot insera direct ferestre in cadrul unui document HTML obisnuit folosind marcajul iframe. Ele se numesc şi *ferestre incluse* (engl.*inline frame*).

Imagini in documentele HTML

Cele mai raspandite formate de imagini intalnite in WWW sunt Graphics Interchange Format – GIF, Joint Photographic Experts Group – JPEG şi Portable Network Graphics – PNG. Ele se folosesc pentru stocarea imaginilor in forma compresata. De remarcat ca HTML nu specifica tipul de format al imaginilor. Acesta este tratat exclusiv de programul navigator, imaginile fiind dpdv al HTML doar date externe.

Pentru includerea imaginilor in paginile WWW se foloseste marcajul IMG. Atributele: SRC pentru specificarea fisierului care conţine imaginea, ALIGN (cu valorile left, right, top, middle sau bottom) pentru specificarea tipului de aliniere, ALT pentru specificarea unei descrieri textuale a imaginii, BORDER pentru specificarea grosimii marginii imaginii, HEIGHT şi WIDTH pentru specificarea inaltimii şi largimii imaginii in numar de pixeli, etc.

Exemplu:

```
<IMG SRC="sunset.jpg" HEIGHT="300" WIDTH="400" ALT="Apus"/>
```

Pentru crearea de legaturi intre imagini sau portiuni de imagini şi actiuni sau resurse se folosesc *hartile de imagini* (engl.*image map*). Actiunile sunt portiuni de cod scripting iar resursele se specifica prin URL. Hartile de imagini se pot rezolva la server sau la client. Rezolvarea presupune gasirea resursei sau actiunii corespunzatoare imaginii sau portiunii de imagine care a fost selectata. Asocierea unei imagini cu o harta se face cu atributul USEMAP pentru rezolvarea la client şi cu atributul ISMAP pentru rezolvarea la server.

Formulare HTML şi comunicare HTTP

Interactiunea dintre CGI şi formularele HTML se face astfel:

- *Formularele* (engl.*form*) furnizeaza paginilor WWW suport pentru interactivitate. Ele se specifica cu elementul form. Acesta accepta atribute ca: *action* pentru a indica programul de prelucrare a datelor formularului, method pentru specificarea metodei HTTP folosita pentru transferul datelor către server (GET sau POST), target pentru a specifica fereastra care conţine formularul, etc.
- In interiorul elementului form se introduc elemente pentru descrierea controalelor de interactivitate. Acestea pot fi:
  - input, un element general cu tipul descris prin atributul type. Acesta poate fi: text pentru o caseta de editare, password pentru o caseta de editare a unei parole, checkbox pentru o caseta de validare, radio pentru un buton de selectie exclusiva (engl.*radio button*), submit pentru un buton de trimitere a datelor către server, etc.
  - select, pentru definirea unui meniu. Optiunile meniului se descriu cu elementele option şi optgroup. Meniul poate fi cu selectie simpla sau multipla (multiple)
  - textarea, pentru definirea unei camp de editare multilinie.
- Pentru vizualizarea datelor trimise către server se va implementa formularul folosind metoda GET şi se va vizualiza URL-ul generat in campul de adresa al navigatorului WWW.

5.2.2. Tehnologii DynamicHTML

Termenul de *programare pe partea de client* (engl.*client side programming*) se refera la executarea de programe la client in scopul cresterii gradului de interactivitate al paginilor WWW. Programele destinate a fi executate la client se pot transmite de la server către client fie in *format sursa*, fie in *format obiect*.

Termenul de *programare pe partea de server* (engl.*server-side programming*) se refera la executarea de programe la server in scopul de a genera date şi/sau rapoarte care sunt trimise clientului sau mai general, pentru extinderea funcţionalitatii unui server de WWW. Datele şi rapoartele sunt trimise clientului de obicei in format HTML.

Termenul *Dynamic HTML* – DHTML nu se refera la o anumita versiune sau facilitate a HTML. El desemneaza acele facilitati din diversele variante sau extensii ale HTML care ajuta la crearea de conţinut dinamic.

Cele mai populare elemente ale DHTML sunt:

- *Foile de stil* (engl.*Cascading Style Sheets* - CSS).
- *Scripting*-ul la client (elementul SCRIPT). Un exemplu este JavaScript. Se refera la incorporarea in cadrul unei pagini a unor programe in format sursa. Ele vor fi executate la client.
- *Obiectele* (elementul OBJECT). Se refera la incorporarea in cadrul unei pagini a unor programe in format obiect. Ele vor fi executate la client.
- *Modelul obiectelor document* (engl.*Document Object Model* – DOM). Acesta este liantul dintre elementele anterioare şi limbajul de marcare HTML sau XML.
- Eventuale mecanisme particulare specifice programului navigator.

Se recomanda separarea *prezentarii* (margini, culori, fonturi) de *conţinutul şi structura* documentului (antet, pagini, paragrafe, titluri, sectiuni). Pentru aceasta se pot folosi *foi de stil* (engl.*Cascading Style Sheets* – CSS).

Prezentarea unui document este in general determinata de mai multi factori:

- Intentiile proiectantului paginii
- Reguli generale pe care trebuie sa le urmeze proiectantul
- Preferintele utilizatorului
- Limitarile cauzate de terminalul pe care se vizualizeaza prezentarea

Din acest motiv apare necesitatea utilizarii mai multor foi de stil – *cascadarea foilor de stil (CSS)*. CSS specifica *stilul de prezentare* al unui document HTML. Exista trei modalitati de a mixa CSS şi HTML: i) in cadrul unui element HTML; ii) in antetul unui document HTML; iii) intr-un fisier separat:

- Specificarea stilului intr-un element individual se face cu atributul STYLE:

```
<HTML>
 <HEAD>
  <TITLE>
   Specificarea stilului in cadrul unui element HTML
  </TITLE>
 </HEAD>
 <BODY>
  <P STYLE = "font-size: 20pt">Text</P>
  <P STYLE = "font-size: 20pt; color: #0000FF">Text</P>
 </BODY>
</HTML>
```

- Specificarea stilului in antet se face cu elementul STYLE:

```
<HTML>
 <HEAD>
  <TITLE>Specificarea stilului in antet
  </TITLE>
  <STYLE TYPE = "text/css">
   EM { background-color: #8000FF; color: white }
   H1 { font-family: Arial, sans-serif }
   P  { font-size: 18pt }
   .blue { color: blue }
  </STYLE>
 </HEAD>
 <BODY>
  <H1 CLASS = "blue">Un antet</H1>
  <P>Un paragraf</P>
  <H1>Alt antet</H1>
  <P CLASS = "blue">Alt <EM>paragraf</EM></P>
 </BODY>
</HTML>
```

- Foaie de stil intr-un fisier separat:

```
EM { background-color: #8000FF; color: white }
H1 { font-family: Arial,
     sans-serif }
P  { font-size: 18pt }
.blue { color: blue }
```

- O foaie de stil pastrata intr-un fisier separat se leaga la un document HTML folosind elementul LINK.

```
<HTML>
 <HEAD>
  <TITLE>
   Importul unei foi de stil
   <LINK REL="stylesheet"
    TYPE="text/css"
    HREF="css3.css"/>
  </TITLE>
 </HEAD>
 <BODY>
  <H1 CLASS = "blue">Un antet</H1>
  <P>Un paragraf</P>
  <H1>Alt antet</H1>
  <P CLASS =
   "blue">Alt<EM>paragraf</EM></P>
 </BODY>
</HTML>
```

Reguli, selectori şi declaratii

O foaie de stil este constituită dintr-o multime de *reguli* care se aplica unui document HTML in scopul generarii unei prezentari a documentului.

O regula de stil conţine o *parte de conditii* (stanga) numita şi *selector* şi o *parte de actiuni* (dreapta) numita şi *declaratie*. Exemplu: P {color:green}, unde:

- P este un selector
- {color:green} este o declaratie. O declaratie poate conţine mai multe asignari
- color:green este o asignare
- color este o proprietate
- green este o valoare

Stilul unui document este o multime de asignari de valori unor variabile numite *proprietati*. O proprietate exprima o calitate sau caracteristica pe care o poate avea un element al unui document HTML.

Asignarea de valori proprietatilor se bazeaza pe mecanismul de *mostenire cu suprascriere* (engl.*overriding inheritance*).

Un document HTML este structurat sub forma unui arbore de elemente carora li se asociaza proprietati cu valori. O pereche proprietate – valoare se transmite prin mostenire de la nivelurile superioare in arbore către frunze. La intalnirea unei noi valori pentru aceiasi proprietate, noua valoare o va suprascrie pe cea veche, fiind considerata mai specifica.

Selectorii regulilor pot fi: tipurile elementelor, atributele elementelor (elementele CLASS şi ID au fost introduse pentru a fi folosite cu CSS), contextul elementelor, respectiv informaţii externe documentului. Pentru detalii se va consulta specificatia CSS1 (CSS de nivel 1).

Moştenirea şi rezolvarea conflictelor în DHTML

In general unui document i se aplica mai multe foi de stil:

- Foaia de stil specifica documentului
- Foaia de stil implicita a programului navigator. Ideal, navigatorul ar trebui sa permita setarea foii de stil implicite in funcţie de preferintele utilizatorului, intr-un mod independent de navigator
- O eventuala foaie de stil externa, specifica tipului de document

In cazul in care mai multe reguli sunt aplicabile unui element, selectia regulii aplicabile se face pe baza unei *strategii de rezolvare a conflictelor*. Conflictele sunt de doua tipuri:

- Conflicte in cadrul aceleiasi foi de stil
- Conflicte intre regulile unor foi de stil diferite

Rezolvarea conflictelor se face astfel:

- Se determina toate regulile aplicabile din toate foile de stil.
- Se ordoneaza regulile selectate de pasul anterior astfel incat regulile marcate ca importante au prioritate.
- Se ordoneaza regulile selectate de pasul anterior in funcţie de originea foii de stil: navigator, utilizator, proiectant (de la proiectant au precedenta maxima). După acest pas vor ramne doar reguli din aceeasi foaie de stil.
- Se ordoneaza regulile selectate de pasul anterior in funcţie de specificitate. O regula este cu atat mai specifica cu cat selectorul ei exprima o conditie mai complexa.
- Se ordoneaza regulile selectate de pasul anterior in funcţie de ordinea in care au fost specificate. Regulile ce apar mai tarziu in document au o prioritate mai mare. Regulile importate din foi de stil externe se considera aflate inainte oricaror reguli prezente explicit in foaia de stil.

Cateva exemple de reguli in foile de stil sunt :

– Selector bazat pe tipul elementelor:
H1 {color: blue}
H1, H2: {color: blue}
– Selector bazat pe clasa (atributul CLASS) elementelor:
.IMPORTANT {color: red}
– Selector bazat pe ID-ul elementelor:
#COPYRIGHT {font-size: small}
– Selector contextual:
LI P {margin-top: 0mm}
TABLE .SMALL P {font-size: small}
– Selectori bazati pe informaţii externe documentului
A:LINK {color: green}
A:ACTIVE {color: red}
A:VISITED {color: blue}
P.INITIAL:FIRST-LETTER {font-size: 200%; float: left}
P.INITIAL:FIRST-LINE {text-transform: uppercase}
– Combinarea tipurilor de selectori
TABLE P.INITIAL:FIRST-LETTER {font-size: 200%; float: left}

Proprietatile din cadrul declaratiilor pot fi:

- Referitoare la fonturi
- Referitoare la spatiere
- Referitoare la imagini
- Referitoare la culori şi fundaluri

5.2.3. Miniaplicatii Java

O *miniaplicatie* (engl.*applet*) Java este un program Java care ruleaza sub controlul unui program client de navigare WWW. Miniaplicatia este transferata de la server către client sub forma de cod obiect (*class* sau *jar*).

Conditia ca o miniaplicatie sa poata fi executata pe calculatorul clientului este ca programul de navigare sa dispuna de un subprogram de interpretare a codului binar al masinii virtuale Java. Un astfel de program navigator se numeste *Java-enabled*.

Restrictii:

i) o miniaplicatie nu se poate atinge (in citire sau scriere) de discul calculatorului client pe care ruleaza; din acest motiv se spune metaforic ca ruleaza *inside the sandbox*;

ii) ii) lansarea in executie a unei miniaplicatii poate dura deoarece descarcarea fisiereleor *class* şi apoi incarcarea lor sub controlul programului client de navigare poate consuma un timp semnificativ de mare.

O miniaplicatie se include intr-o pagina WWW cu elementele APPLET sau OBJECT. Initial s-a folosit elementul APPLET, deoarece singurele programe executabile care se puteau include in paginile WWW sub forma de cod obiect erau miniaplicatiile Java. Exemplu:

```
<APPLET CODE="Miniaplicatie" CODEBASE="." WIDTH="300" HEIGHT="200">
</APPLET>
```

Ulterior s-a introdus elementul OBJECT pentru a se permite includerea şi a altor aplicatii sub forma de cod obiect in paginile WWW. Exemplu:

```
<OBJECT WIDTH="500" HEIGHT="300"> <PARAM NAME="code" VALUE=
"Miniaplicatie"/>

  <PARAM NAME="codebase" VALUE ="."/>

</OBJECT>
```

Construirea unei miniaplicatii Java presupune urmatoarele:

- O miniaplicatie Java este o aplicatie grafica care foloseste bibliotecile Java pentru construirea de intrft grafice *AWT* sau *Swing*.
- O miniaplicatie nu are constructor deoarece este creata de programul de navigare sub controlul caruia ruleaza. Ea se construieste prin extinderea clasei Applet din AWT sau JApplet din Swing şi redefinirea urmatoarelor metode (cel putin init() trebuie redefinita):
  - init(), se executa o singura data la incarcarea miniplicatiei.
  - start(), se executa după init() şi ori de cate ori este reafisata pagina care conţine miniaplicatia
  - stop(), se executa cand pagina ce conţine miniaplicatia nu mai este afisata şi imediat inainte de destroy()
  - destroy(), se executa cand miniaplicatia se inchide şi trebuie descarcata din memorie
- Operatiile din cadrul redefinirii metodei init() sunt asemenatoare cu cele din cadrul unui constructor al unei aplicatii grafice obisnuite. Spre exemplu, aici se initializeaza controalele grafice membre ale interfetei miniaplicatiei şi se adauga la miniaplicatie. Din acest motiv orice miniaplicatie poate fi rulata şi ca o aplicatie obisnuita dacă se defineste o metoda main() in care se creeaza un obiect miniaplicatie şi apoi se executa operatiile din init().

Interfete grafice in Java

Se spune ca AWT (java.awt) este de *categorie grea* (engl.*heavyweight*), iar Swing (javax.swing) este de *categorie usoara* (engl.*lightweight*). AWT se bazeaza pe obiectele grafice ale platformei gazda in timp ce Swing redeseneaza toate controalele grafice. Astfel AWT este mai rapida, dar dependenta de platforma (la aspect) in timp ce Swing este mai lenta, dar independenta de platforma.

O interfata grafica este compusa din *componente*. Componentele pot fi amplasate in *containere*, care sunt la randul lor componente. Astfel se pot construi interfete grafice complexe. Modelul de programare folosit la programarea interfetelor grafice se numeste *programare orientata pe evenimente (*engl.*event-driven programming*). Componentele genereaza evenimente la actiunile utilizatorului. La componente se inregistreaza niste obiecte speciale numite *obiecte ascultator* (engl.*listener*) interesate sa primeasca aceste evenimente.

Obiectele ascultator primesc evenimentele sub forma unor apeluri de metode. Din acest motiv ele trebuie sa implementeze anumite interfete astfel incat sa permita aceste apeluri. Aceste interfete sunt subclase ale clasei de baza EventListener.

In momentul in care un obiect ascultator receptioneaza un eveniment, in apelul corespunzator evenimentului se transmit şi informaţii referitoare la obiectul grafic care a transmis evenimentul.

5.2.4. Programare la client folosind *scripting*

Prin limbaj de s*cripting* se intelege un limbaj de programare interpretat. Solutia de interpretare permite transmiterea programelor in cod sursa, cu conditia existentei la destinatie a unui interpretor pentru executia lor. Exemple de limbaje de scripting: dBase, Visual Basic, Perl, Unix shell, Python, Tcl, Ruby, etc. Un program scris intr-un limbaj de scripting se numeste *script*.

In WWW, scripting-ul se foloseste atat la client cat şi la server. Executia unui *script* la server poate produce conţinut ce este transmis prin HTTP clientului (de exemplu prin CGI). Executia unui script la client produce conţinut dinamic. Scripting-ul la client este parte a tehnologiei *Dynamic HTML – DHTML*.

Deosebirea esentiala dintre un limbaj de programare compilat şi un limbaj de scripting interpretat este *momentul legarii* (engl.*binding*) - momentul in care devin cunoscute atributele unei variabile. La limbajele compilate momentul legarii este cel al traducerii, iar la cele interpretate este cel al executiei. Amanarea legarii atributelor conduce la flexibilitate a executiei in dauna eficientei.

JavaScript este un *limbaj de scripting orientat pe obiect* inventat de Netscape. Spre deosebire de limbajele orientate pe obiect bazate pe clase, cum sunt C++ şi Java, JavaScript este un *limbaj orientat pe obiect bazat pe prototipuri*.

JavaScript este un limbaj *extensibil*. Exista o *componenta de baza*, numita *Core JavaScript*, care conţine elementele de baza ale limbajului (cuvinte cheie, operatori, enunturi, structuri de control, modelul obiectual) şi o multime de obiecte de baza (ex.*Array*, *Date*, *Math*). La ea se pot adauga extensii sub forma unor obiecte suplimentare. Exemple sunt *Client-side JavaScript* care conţine in plus obiecte pentru controlul programului navigator şi DOM şi *Server-side JavaScript* care conţine obiecte suport pentru partea de server (de ex. pentru comunicarea cu o baza de date relationala). In conţinuare prin JavaScript vom subintelege partea client a JavaScript.

Navigatoarele WWW interpreteaza scripturile JavaScript din paginile HTML. Ele citesc pagina, interpreteaza marcajele şi o afiseaza, executand in acelasi timp scripturile JavaScript pe masura intalnirii lor in cadrul paginii. Rezultatul acestui proces de interpretare/executie este vizualizat de utilizator in fereastra navigatorului.

Intre JavaScript şi Java exista şi asemanari şi deosebiri fundamentale. Asemanarile se refera la sintaxa enunturilor şi a structurilor de control. Deosebirile: Java are *legare la compilare* (engl.*statically typed*), este *puternic tipizat* (engl.*strongly-typed*) şi foloseste un *model obiectual bazat pe clase*. JavaScript are *legare la executie* (engl.*dynamically typed*), este mult mai permisiv in ceea ce priveste declaratiile (engl.*loosely-typed*) şi foloseste un *model obiectual bazat pe prototipuri*.

Utilizarea JavaScript intr-o pagina HTML presupune folosirea elementului *SCRIPT*:
<SCRIPT> *Enunturi JavaScript...* </SCRIPT>

Specificarea limbajului şi versiunii de scripting se fac cu atributul *LANGUAGE*:
<SCRIPT LANGUAGE="JavaScript1.3">

Specificarea codului JavaScript se poate face direct in elementul *SCRIPT*:

```
<HTML>
 <HEAD>
  <SCRIPT LANGUAGE ="JavaScript1.3">
   <!-- Ascunde scriptul pentru navigatoarele ce nu stiu Javascript.
     document.write("Salut !");
   // Sfarsitul scriptului. -->
```

```
  </SCRIPT>
 </HEAD>
</HTML>
```

Specificarea codului JavaScript se poate face şi intr-un fisier/URL separat:

```
<HTML>
 <HEAD>
  <SCRIPT SRC="js1.js"></SCRIPT>
 </HEAD>
</HTML>
```

Utilizarea *entitatilor JavaScript* permite specificarea de expresii JavaScript ca valori ale atributelor elementelor HTML (nu merge la IE !?). O entitate JavaScript are sintaxa &{*expresie JavaScript*}; Un exemplu de folosire:

```
<SCRIPT LANGUAGE="JavaScript">
 var lungime = 25;
</SCRIPT>
<HR WIDTH="&{lungime*2};%" ALIGN="left"/>
```

Sirurile de caractere in cadrul unui literal JavaScript se pun intre apostroafe (merge şi \" in IE):

```
document.writeln("<HR ALIGN='left' WIDTH=" + lungime*2 + "%>")
```

Aplicatiile JavaScript sunt *orientate pe evenimente* (engl.*event-driven*). Un eveniment este de obicei rezultatul unei actiuni a utilizatorului. Un script poate reactiona la evenimente prin intermediul unui secvente de cod de tratare a evenimentului (engl.*event handler*). Un event handler este o secventa de cod JavaScript, de obicei un apel de funcţie JavaScript.

Dacă un eveniment se aplica unui element HTML, atunci se poate specifica un event handler in cadrul elementului. Evenimentul se aplica obiectului JavaScript creat de element.

<*element* on*Eveniment*="*cod JavaScript*">

Exemplu:

```
<HEAD>
 <SCRIPT>
  funcţion calcul(f) {
     f.rez.value = eval(f.expr.value)
  }
 </SCRIPT>
</HEAD>
<BODY>
 <FORM>
  Introduceti o expresie:
  <INPUT TYPE="text" NAME="expr" SIZE=15/>
  <INPUT TYPE="button" VALUE="Calculeaza" onClick="calcul(this.form)"/><BR/>
  Rezultat:
  <INPUT TYPE="text" NAME="rez" SIZE=15/>
 </FORM>
</BODY>
```

Obiectul this desemneaza obiectul buton, iar this.form formularul care conţine butonul. Funcţia calcul() primeste ca argument un formular, preia expresia de calculat din campul expr, o evalueaza cu eval() şi depune rezultatul in campul rez. Evenimentul tratat este *Click* pe buton.

Core JavaScript

Variabilele se declara la initializare sau optional cu var. Pot fi *globale* sau *locale* (definite in corpul unei funcţii). Cele locale se declara obligatoriu cu var.

Exista tipuri primitive şi tipuri compuse. *Tipurile primitive* sunt: numerele (intregi şi reali), valorile logice, sirurile de caractere, valoarea null şi valoarea undefined. *Tipurile compuse* sunt *vectorii* şi *obiectele*.

Pentru specificarea *constantelor* (numite şi *valori*) se folosesc *literali*. Exista literali pentru specificarea numerelor (2, 3.14), valorilor logice (true şi false), vectorilor ([1,2,3,"aaa"]) şi obiectelor ({camp_1:"a",camp_2:3}).

S*tructurile de control* sunt cele din Java: secventierea, if-else, switch, for, while, do-while, label, break şi conţinue. Exista in plus *structurile pentru manipularea obiectelor*: for-in şi with. *Comentariile* sunt ca in Java.

Caramizile de baza ale JavaScript sunt *funcţiile*. Ele se declara astfel:

function *nume*(*argumente*) { *corpul funcţiei* }

Funcţiile pot intoarce optional o valoare cu return şi pot fi recursive. Argumentele funcţiilor sunt disponibile din vectorul arguments. Exista o multime de funcţii predefinite (vezi documentatia de la Netscape).

Funcţiile şi variabilele globale pot fi referite şi din alte ferestre dacă se prefixeaza cu numele ferestrei in care au fost definite.

Relativ la modelul obiectual din JavaScript, se mentioneaza urmatoarele aspecte:

- Construirea unui obiect presupune:
- Initializarea cu un literal:

nume = {$prop_1$:val_1, $prop_2$:$value_2$, ..., $prop_n$:val_n};

- Definirea unei funcţii constructor:

```
funcţion TipObiect(argumente) {
  this.prop1 = val1;
  …
}
obiect = new TipObiect(argumente);
```

- Proprietatile unui obiect se pot accesa cu *obiect.prop* sau cu *obiect*["*prop*"].
- Obiecte JavaScript predefinite: Array, Boolean, Date, Funcţion, Math, Number, RegExp, String.
- *Ierarhii de obiecte* in JavaScript: dacă se doreste crearea unui obiect care mosteneste de la un alt obiect se seteaza valoarea prototype a sa la obiectul de la care se mosteneste. Notiunea de *clasa de baza* din limbajele bazate pe clase a fost inlocuita cu notiunea de *obiect prototip*.

Concluzii:

- Pe cat posibil incercati sa separati partea de conţinut de partea de prezentare din pagini. Pentru aceasta identificati intai unitatile de conţinut pe care trebuie sa le conţina pagina şi abia apoi puneti-va problema modului de prezentare al acestora.
- Aveti in vedere ca descarcarea unor programe in cod obiect şi rularea lor in programul navigator poate duce la incetinirea incarcarii paginii.
- Folositi scriptingul la client numai atunci cand este absolut necesar. Scriptingul nu face sa creasca dependenta paginilor pe care le scrieti de programul navigator.
- Atunci cand doriti sa creati pagini complexe şi folosirea scriptingului la client este absolut

necesara, incercati sa identificati cu clientul dumneavoastra navigatoarele carora le sunt adresate paginile. Un pas important va fi apoi testarea navigatorului inainte de afisarea paginilor sau a anumitor parti specifice din pagini.

5.3. PROGRAMARE PE PARTEA DE SERVER

5.3.1. Servere WWW

Un *server WWW* este un program cu rol de server care este capabil sa raspunda la cereri HTTP. Modul de funcţionare al unui server WWW este foarte important pentru infrastructura şi aplicatiile WWW, desi in acelasi timp este ascuns utilizatorilor uzuali ai WWW. Probleme importante referitoare la serverele WWW sunt:

- Performanta
- Configurarea şi administrarea
- Extensia şi programarea

Un exemplu de server WWW foarte folosit şi disponibil liber in domeniul public este Apache. Un sondaj din 1998 a aratat ca 51.9% din toate serverele active de WWW erau servere Apache.

Relativ la performantele serverelor WWW, trebuie precizate următoarele aspecte:

- La nivelul anului 1999 se mentioneaza in literatura doua suite (pachete de programe) standard (engl.*benchmark*) pentru testarea performantelor serverelor WWW:
  - WebStone, dezvoltata de Silicon Graphics, Inc. – SGI
  - SPECweb96, dezvoltata de Standard Performance Evaluation Corporation – SPEC
- WebStone este o suita configurabila de teste de performanta, destinata a fi utilizata de administratorii WWW. Configurabilitatea este utila pentru a proiecta teste specifice pentru diverse sabloane de acces la servere.
- SPECweb96 este o suita fixa de teste de performanta. Sabloanele de test au fost obţinute analizand cele mai frecvente moduri de utilizare ale serverelor WWW.

  Optiunile de configurare ale unui server WWW se refera in general la:
  - Modul in care se va executa serverul pe masina care il gazduieste.
  - Dacă serverul deserveste mai multe masini gazda şi in caz afirmativ modul in care realizeaza acest lucru. Dacă serverul deserveste mai multe masini gazda atunci acestea se numesc *gazde virtuale* (engl.*virtual host*).
  - Modul de lucru: *server de origine* sau *proxy*.

Executia serverelor WWW poate fi de două tipuri:

- Executie la cerere
  - Este o metoda veche. Presupune existenta unui *superserver* care asculta toate cererile şi pentru fiecare in parte activeaza şi executa serverul corespunzator. In UNIX acest superserver se numeste *inetd*. Aceasta metoda are marele dezavantaj ca necesita prea multe resurse sistem de fiecare data cand se incarca serverul in memorie şi se creaza procesul aferent.
- Executie permanenta
  - Serverul este pornit manual de utilizator sau automat la pornirea sistemului şi se executa in permanenta. După pornire serverul asculta cererile HTTP pe portul pe care a fost configurat. Pornirea manuala este recomandata intr-un mediu de dezvoltare a unei aplicatii WWW. Pornirea automata este recomandata după ce aplicatia WWW a fost instalata (engl.*deployed*).
  - Serverul poate fi pornit de la linia de comanda sau instalat ca serviciu al sistemului de operare.

5.3.2. Gazde virtuale

Un server WWW poate deservi una sau mai multe masini gazda. In al doilea caz masinile gazda deservite se numesc *gazde virtuale*. Gazdele virtuale usureaza activitatea de administrare a serverului deoarece: exista o singura instanta a serverului, exista o singura configuratie a serverului, se monitorizeaza executia unui singur server. Exista doua tipuri de gazde virtuale: *gazde virtuale IP* şi *gazde virtuale non-IP*.

Gazdele virtuale IP:

- Fiecare gazda virtuala este o gazda IP avand o adresa de IP distincta care apare ca intrare in DNS.
- Se asigneaza toate adresele de IP ale gazdelor virtuale deservite masinii pe care ruleaza serverul WWW.
- Metoda este simpla dar are doua dezavantaje: i) este nevoie de o adresa de IP distincta pentru fiecare gazda virtuala; ii) asignarea mai multor adrese de IP unei singure masini poate crea probleme retelei.

Gazdele virtuale non-IP:

- Se bazeaza pe un suport special oferit de protocolul HTTP. Acest suport pentru deservirea gazdelor virtuale non-IP este disponibil numai in HTTP/1.1.
- HTTP/1.1 a introdus in cadrul unei cereri HTTP antetul special HOST. Acest camp antet este obligatoriu. El conţine numele gazdei careia ii este adresata cererea.
- Configurarea gazdelor virtuale deservite se face analog cu cazul anterior. Singura diferenta este ca intrarile DNS ale gazdelor virtuale vor conţine acum aceeasi adresa de IP.
- Metoda are doua avantaje: i) este nevoie de o singura adresa de IP pentru deservirea tuturor gazdelor virtuale; ii) crearea unei noi gazde virtuale non-IP se poate face mai usor decat in cazul gazdelor virtuale IP. Metoda are un mare dezavantaj: se poate aplica numai pentru versiunile HTTP ulterioare HTTP/1.1.

Serverele WWW pot fi:

➢ **Servere de origine**

- Majoritatea serverelor de WWW sunt configurate ca *servere de origine*. In acest caz cererile HTTP sunt tratate local de server.
- O problema importanta este localizarea resurselor, cu alte cuvinte maparea URL-ului intr-o adresa a unei resurse din cadrul sistemului local de fisiere. Resursele sunt de doua tipuri: *statice* (stocate fizic) şi *dinamice* (generate de programe externe).
- Spatiul WWW al unui utilizator specific se refera in URL prin caracterul ~ urmat de numele utilizatorului. Spatiul WWW al utilizatorilor se poate organiza in doua moduri:
  - Crearea unui spatiu WWW central pentru toti utilizatorii. Unui utilizator ii va revni un director in acest spatiu: webhome/name/.
  - Crearea unui director special pentru fiecare utilizator in cadrul directorului personal: /home/name/WWW/.

➢ **Servere proxy**

- Un server proxy accepta cereri pentru resurse şi fie le rezolva din memoria *cache* locala, fie prin inaintarea cererii către serverul de origine.
- Un server proxy actioneaza astfel ca un intermediar intre client şi serverul de origine. Scopul intermedierii pote fi filtrarea cererilor sau trecerea de la un protocol nesigur la un protocol sigur, de exemplu HTTPS.

5.3.3. Tehnologiile SSI şi CGI

SSI (engl.*Server Side Include*) este o tehnologie ce permite includerea de informaţii intr-o pagina WWW inainte de trimiterea paginii la client.

O pagina care foloseste SSI conţine instructiuni speciale. Aceste instructiuni sunt interpretate de server ori de cate ori pagina este ceruta de un client. Instructiunile pot specifica: includerea unor documente in pagina curenta (de exemplu un *header* sau un *footer*), a datei curente, a valorii unui contor de acces, etc.

Avantajul SSI fata de CGI este ca este o tehnologie mult mai simpla şi ca nu implica apelul nici unui program extern.

Nu exista un standard de SSI şi de aceea fiecare server are o sintaxa şi funcţionalitate proprie pentru SSI. Pentru serverul Apache instructiunile SSI sunt de forma unor comentarii HTML/XML cu structura urmatoare:

<!-- #element $atribut_1$=$valoare_1$ $atribut_2$=$valoare_2$ ... -->

Un exemplu este urmatorul:

<!--#include virtual="/header.html" -->

Serverul Apache permite folosirea şi a unor facilitati SSI avansate desemnate prin acronimul XSSI. Un exemplu este posibilitatea definirii unui flux de control ca in programarea procedurala prin comenzile if, elif, else şi endif.

Aplicatiile de comert electronic au o parte semnificativa pe partea de server. Pentru a descrie aceasta parte se foloseste frecvent termenul de *aplicatie WWW* = extensia dinamica a unui server WWW. Aplicatiile WWW sunt in general de doua tipuri:

- *Aplicatii orientate pe prezentare*. Conţin pagini WWW interactive şi sunt capabile sa genereze conţinut dinamic ca raspuns la cererile clientilor.
- *Aplicatii orientate pe serviciu*. Implementeaza un punct terminal pentru un serviciu WWW. *Serviciu WWW* = un serviciu oferit de o aplicatie altor aplicatii, prin intermediul platformei WWW.

Aplicatiile WWW conţin *componente WWW*. Componentele WWW sunt caramizile pe baza carora se poate extinde dinamic funcţionalitatea unui server WWW. In tehnologia Java, componentele WWW sunt miniservere sau pagini JSP.

Componentele WWW sunt suportate de serviciile unei platforme speciale de executie numita *container WWW*. Containerul furnizeaza servicii ca: dispecerizarea cererilor, securitate, concurenta, gestiunea ciclului de viata, etc.

O aplicatie WWW consta in general din: componente WWW, fisiere de resurse statice şi clase/biblioteci de ajutor (engl.*helper*).

Funcţionalitatea standard oferita de un server WWW este insuficienta pentru programarea aplicatiilor de comert electronic. O aplicatie de comert electronic are o arhitectura multistrat, numarul de straturi fiind de obicei minim 3: client, server WWW, server de baze de date. Pe langa serverul WWW mai exista obiectele din domeniul problemei (engl.*business objects*). Impreuna ele formeaza stratul intermediar al unei arhitecturi tipice pentru o aplicatie de comert electronic. In consecinta se pune problema extinderii funcţionalitatii unui server de WWW. O varianta este folosirea interfetei CGI, insa ea prezinta o serie de dezavantaje:

- Pentru fiecare cerere HTTP trebuie startat un proces separat.
- Interfata dintre serverul WWW şi scriptul CGI este nesatisfacatoare.

O abordare eleganta o constituie o solutie Java pentru extinderea funcţionalitatii unui server WWW, şi anume miniserverele Java (engl.*Java servlet*). Aceasta solutie are urmatoarele avantaje:

- Portabilitate

- Miniserverele sunt rezidente şi starea este persistenta intre cereri succesive
- Beneficierea de avantajele tehnologiei Java: orientare pe obiect, modelul de securitate Java, integrarea relativ usoara cu tehnologii ca RMI şi CORBA

CGI (engl.*Common Gateway Interface*) defineste o interfata pentru comunicarea dintre un server de informaţii (cum este cazul unui server WWW) şi un program de aplicatie.

CGI este o interfata independenta de limbaj. Este posibil sa se implementeze o aplicatie CGI (numita şi *script CGI*) in orice limbaj care suporta comunicarea standard intre procese prin intermediul intrarii şi iesirii standard şi a variabilelor de mediu. Pentru scripturile CGI se foloseste deseori unul dintre limbajele: Perl, Python sau Tcl. Se pot folosi insa şi limbaje gen C/C++.

Procesul de comunicare decurge astfel:

- După primirea cererii HTTP şi inaintea pornirii scriptului CGI, serverul initializeaza o multime de variabile de mediu (engl.*environment variables*). Aceste variabile vor fi mostenite de procesul corespunzator lansarii scriptului CGI. Existe variabile independente de cerere şi variabile dependente de cerere. Pe langa aceste variabile de mediu, dacă protocolul este HTTP, se creeaza cate o variabila de mediu ce incepe cu prefixul HTTP pentru fiecare linie din antetul cererii (de exemplu HTTP_ACCEPT).
- Dacă cererea conţine informaţii aditionale după antetul cererii, atunci informaţia este trimisa scripului CGI la intrarea standard. Se trimite un numar de octeti egal cu valoarea variabilei de mediu CONTENT_LENGTH.
- Scriptul genereaza datele de iesire la iesirea standard. Pentru generarea raspunsului către client, serverul fie interpreteaza şi prelucreaza aceste date, fie le inainteaza nealterate. El indica acest acest lucru serverului prin antete speciale.

Funcţionarea miniserverelor Java presupune urmatorii pasi:

- Un program client emite o cerere HTTP către un server WWW.
- Serverul interpreteaza cererea şi executa o secventa de program careia ii transmite parametrii cererii.
- Programul apelat interpreteaza parametrii cererii şi executa o portiune de cod care genereaza dinamic o pagina HTML.

Prelucrarile executate de miniserver pot duce la:

- Generarea unei pagini WWW statice.
- Generarea unei pagini WWW modificata prin inserarea unui conţinut dinamic.
- Configurarea unei pagini WWW pe baza parametrilor cererii HTTP. Parametrii sunt preluati de la utilizator printr-un formular HTML.

In general miniserverele sunt potrivite pentru aplicatiile orientate pe serviciu sau pentru controlul aplicatiilor orientate pe prezentare.

5.3.4. Miniservere Java

La fel ca şi o miniaplicatie, un miniserver nu conţine o funcţie main(). El va fi invocat de către containerul de miniserveri la receptionarea unei cereri HTTP de către serverul WWW. Pentru aceasta un miniserver trebuie sa implementeze o interfata javax.servlet.HttpServlet

Dezvoltarea unui minserver presupune extinderea clasei javax.servlet.HttpServlet şi redefinerea metodelor sale. Interfata cuprinde metodele service(), init() şi destroy().

La executarea unei cereri către un miniserver au loc urmatoarele: i) dacă miniserverul nu exista atunci containerul de miniserveri incarca clasa corespunzatoare miniserverului, creaza o instanta a acestei clase şi apeleaza metoda init(); ii) se apeleaza metoda service(). In funcţie de tipul cererii, ea va apela o metoda do*YYY*() unde *YYY* identifica cererea HTTP, de exemplu

doGet() sau doPost().

Cand containerul trebuie sa descarce miniserverul din memorie el va apela metoda destroy().

Metodele init() şi destroy() se refera la ciclul de viata al miniserverului (la fel ca la miniaplicatii). Aceste operatii se executa mult mai rar decat service().

Prelucrarile din cadrul unui miniserver se fac astfel:

- In cadrul metodelor init() şi destroy() se implementeaza acele prelucrari care asigura persistenta informaţiei stocate de miniserver.
- In metodele service() sau do*YYY*() se implementeaza prelucrarile legate de tratarea cererilor primite de la client. Acestea presupun de obicei urmatoarele:
- Setarea tipului conţinutului in obiectul raspuns HTTP, de tip javax.servlet.http.HttpServletResponse:

```
raspuns.setContentType("text/html");
```

- Obţinerea unui flux de iesire la nivel de octet (OutputStream obţinut din raspuns cu getOutputStream()) sau la nivel de caracter (PrintWriter obţinut din raspuns cu getWriter()):

```
PrintWriter iesire = raspuns.getWriter();
```

- Obţinerea valorilor parametrilor cererii setati de client in formularul HTML, din obiectul cerere de tip javax.servlet.http.HttpServletResponse, cu getParameter(). Dacă numele parametrilor este necunoscut atunci se poate obţine lista numelor parametrilor cu getParameterNames().
- Construirea raspunsului prin scrierea in fluxul de iesire a unor enunturi HTML.
- Pentru aflarea de informaţii de configurare se foloseste metoda getServletConfig() care intoarce un obiect SevletConfig. Din el se pot extrage parametrii de initializare ai miniserverului cu getInitParameter() şi getInitParameterNames().

Obiecte partajate:

- Componentele WWW in general şi miniserverele in particular folosesc de obicei şi alte obiecte pentru a-şi indeplini sarcinile. Astfel: i) pot folosi obiecte private; ii) pot folosi obiecte publice – atribute ale unui domeniu standard; iii) pot accesa baze de date; iv) pot accesa alte resurse WWW.
- Componentele WWW cooperante pot partaja informaţii sub forma unor obiecte definite ca atribute ale unui domeniu standard: *context WWW* (*aplicatie*), *sesiune*, *cerere HTTP* şi respectiv *pagina JSP*.
- Pentru reprezentarea acestor patru domenii, in pachetul javax.servlet sunt definite patru clase: ServletContext, HttpSession, ServletRequest şi JspContext.
- Accesul la aceste obiecte se face prin metode de tip [get|set]Attribute.

Fire multiple in miniservere:

- Containerul de miniserveri utilizeaza fire de executie separate pentru tratarea cererilor. Pentru aceasta containerul are o rezerva (engl.*pool*) de fire de executie. Fiecarei cereri i se aloca un fir.
- Deoarece este teoretic posibil sa se execute cereri simultane pentru un acelasi miniserver, se pune problema gestionarii corecte a resurselor miniserverului ce sunt partajate de aceste cereri multiple. Resursele partajate pot fi: atribute ale domeniilor standard, variabile membre ale clasei miniserverului, fisiere, baze de date, conexiuni de retea, etc.
- Din acest motiv controlul accesului la resursele partajate se va realiza folosind tehnici de sincronizare.

- Variante:
  - Cea mai simpla solutie este ca metoda service() sa se implementeze cu synchornized.
  - Dacă insa sectiunea critica din cadrul acestei funcţii se executa conditionat, atunci este mai eficient sa se foloseasca synchronized doar pentru portiunea de cod corespunzatoare acelei sectiuni critice.
  - Se poate crea o clasa pentru accesul la resursa partajata, metodele de acces fiind declarate synchronized.

Gestiunea sesiunii de lucru:

- O problema majora a HTTP este ca nu permite pastrarea starii comunicarii dintre un client şi un server WWW. Pentru a inlatura aceasta problema s-au propus diverse metode:
- *Folosirea campurilor ascunse*. Aceasta metoda presupune transmiterea repetata de la server către client şi apoi de la client către server a istoricului interactiunii sub forma unei multimi de campuri ascunse intr-un formular HTML.Un exemplu de camp ascuns este <INPUT TYPE="hidden" name="ascuns1" value="element1">. De fiecare data cand utilizatorul selecteaza un nou element serverul primeste o pereche nume-valoare de forma element_selectat="…", după care serverul creaza un nou camp ascuns pentru acest element şi il paseaza in formularul raspuns, etc
- *Rescrierea URL-ului*. Aceasta metoda consta in adaugarea de informaţii suplimentare la un URL pentru a identifica in mod unic o sesiune.
- *Folosirea autentificarii HTTP*. Aceasta metoda presupune identificarea şi autentificarea utilizatorului la primul sau acces asupra serverului. Ulterior serverul va identifica utilizatorul din antetele HTTP.
- *Folosirea cookie-urilor*.Un cookie este o informaţie pe care un server WWW o poate stoca pe calculatorul unui client. Ulterior, aceasta informaţie va fi trimisa serverului la fiecare cerere a clientului. Serverul poate folosi aceasta informaţie pentru gestiunea sesiunii.

Gestiunea sesiunii de lucru folosind miniservere:

- Un *cookie* se creaza şi se adauga la raspunsul HTTP inaintea setarii tipului conţinutului.
  Cookie c = new Cookie("element_selectat", "carte1");
  raspuns.addCookie(c);
- Serverul va determina ce *cookie*-uri exista in cererea HTTP cu getCookies().
- O *sesiune* consta in una sau mai multe cereri HTTP către un server WWW de-alungul unei perioade de timp.
- Implementarea unei sesiuni foloseste clasa HttpSession. Un obiect sesiune este pastrat pe server şi el stocheaza informaţiiile despre un client de-alungul unei sesiuni a clientului respectiv.
- Clasa HttpSession foloseste clasa Cookie pentru stocarea identificatoului sesiunii la client sub forma unui *cookie*. Sesiunea este pastrata un interval maxim de timp intre doua cereri succesive ale clientului.
- Informaţia se pastreaza in obiectul sesiune sub forma unor perechi variabila-valoare. Pentru accesul la aceste perechi se folosesc metodele setAttribute(), removeAttribute(), getAttribute(), getAttributeNames();

Dezvoltarea unei aplicatii care foloseste miniservere presupune următoarele:

- Se compileaza sursele *java* pentru a obţine fisierele *class*.
- Se creaza un subdirector in directorul *webpps*. Acest subdirector conţine pagina radacina a aplicatiei şi se numeste *context*.
- Se creaza un subdirector *WEB-INF* care va conţine:
  - un fisier de configurare a aplicatiei *web,xml*

- un subdirector *classes* unde se depun fisierele *class*, pe subdirectoare corespunzatoare structurii de pachete a programului *java*

- Apelul miniserverului se poate face direct, dintr-un formular sau dintr-o legatura. In toate cazurile se va specifica un URL de forma:

  URLGazda/context/servlet/fisier_class

  Un exemplu este:

  http://localhost/Miniserver1/servlet/Miniserver1

- Un miniserver poate fi repornit fie prin repornirea containerului de miniservere, fie doar prin repornirea lui individuala.

JSP (engl.*Java Server Pages*) este o tehnologie pentru generarea de pagini dinamice bazata pe ideea mixarii de cod Java cu cod HTML in paginile WWW. Este recomandata pentru aplicatiile WWW orientate pe prezentare.

O alta tehnologie de pagini dinamice foarte folosita la ora actuala este PHP. PHP este un limbaj de scripting pentru partea de server, inspirat din C. Codul PHP poate fi incorporat in paginile WWW.

JSP funcţioneaza peste o arhitectura de miniservere. La incarcarea unui JSP, se genereaza automat codul *java* pentru miniserverul corespunzator şi apoi acesta este compilat şi incarcat in containerul de miniserveri. Acest proces se repeta ori de cate ori codul JSP este modificat.

Elemente de baza ale JSP:

- In JSP se pot referi niste obiecte predefinite: HttpServletRequest request, HttpServletResponse response, jsp.PageContext pageContext, http.HttpSession session, ServletContext application, jsp.JspWriter out, ServerConfig config, java.lang.Object page.
- Exista trei tipuri de elemente de scripting in JSP:
- *Expresii* de forma <%= *expresie* %>
- *Miniscripturi* (engl.*scriptlet*) de forma <% *cod Java* %>
- *Declaratii* de forma <%! *Cod* %>
- Expresiile se utilizeaza pentru a insera valori direct in pagina generata. De exemplu: Current time: <%= new java.util.Date() %>
- Miniscripturile reprezinta un cod Java mai complex decat niste expresii simple.
  <% String queryData = request.getQueryString();
  out.println("Date GET:" + queryData); %>
- Declaratiile permit inserarea de metode sau campuri in codul *java* al corpului clasei miniserver, in afara metodei _jspService().
  <%! private int jj =10; %>

5.4. SERVERE DE BAZE DE DATE

Bazele de date relationale bazate pe SQL sunt cele mai raspandite sisteme comerciale de baze de date. Prin *baza de date relationala* vom intelege o multime de relatii (tabele) accesate prin limbajul de specializat SQL.

Un *tabel* sau *relatie* poate fi gandit ca o reprezentare bidimensionala a unei colectii de date, structurata sub forma unei multimi de linii şi coloane. O *linie* corespunde unei *inregistrari* şi o *coloana* corespunde unui *camp*. Fiecare camp are un *nume* şi un *tip*. Dacă un camp conţine o valoare unica atunci el se numeste *cheie primara* (engl.*primary key*). Se pot crea legaturi intre

relatii prin stocarea cheilor primare ale unei relatii intr-o alta relatie sub forma de *chei externe* (engl.*foreign key*).

SQL (engl.*Structured Query Language*) este un limbaj pentru definirea, interogarea şi actualizarea bazelor de date relationale. Cateva comenzi SQL foarte folosite sunt select, insert, delete, update, commit, rollback, grant şi revoke.

Comenzile commit şi rollback sunt utile pentru lucrul cu tranzactii. O *tranzactie* este o secventa de operatii de acces la o baza de date, secventa ce are proprietatea de *atomicitate*. Acest lucru inseamna ca fie secventa nu se executa, fie trebuie executata in totalitate pentru pastrarea consistentei datelor.

O baza de date care lucreaza in regim tranzactional se poate afla in doua stari:

- starea de *autovalidare* (engl.*autocommit*), caz in care orice cerere de actualizare declanseaza automat modificarea conţinutului bazei de date.
- starea de *validare manuala* (engl.*manual commit*), caz in care modificarea conţinutului bazei de date are loc atunci cand se executa comanda commit. Pana la executarea acestei comenzi, starea bazei de date dinaintea tranzactiei poate fi refacuta cu comanda rollback.

Comenzile grant şi revoke permit administratorilor de sistem sa creeze utilizatori şi sa le dea respectiv ia drepturi pe urmatoarele patru nivele:

- Nivel global: drepturile globale se aplica la toate bazele de date de pe un anumit server
- Nivelul baza de date: drepturile alocate la acest nivel se aplica tuturor relatiilor unei baze de date.
- Nivelul relatie: drepturile alocate la acest nivel se aplica tuturor coloanelor unei relatii.
- Nivelul coloana: dreptuile la acest nivel se aplica individual, coloanelor unei relatii.

Funcţiile unui server de baze de date sunt :

- Interpretarea cererilor SQL primite de la clienti, executia lor şi returnarea rezultatelor.
- Optimizarea interogarilor. O interogare SQL se poate executa in mai multe moduri, diferentele intre timpii de raspuns corespunzatori fiecarui mod putand fi mari. Un server va analiza cererea SQL, va examina modul de stocare a relatiilor şi va produce intr-un timp rezonabil un plan de executie cat mai eficient.
- Prevenirea erorilor datorate unui acces concurent la date.
- Detectarea şi inlaturarea situatiilor de interblocaj.
- Administrarea accesului controlat la date din motive de securitate.
- Administrarea operatiilor de arhivare/restaurare a bazei de date. Restaurarea bazei de date in cazul unor caderi ale sistemului se realizeaza prin gestiunea unui *jurnal* (engl.*log*) al tuturor tranzactiilor executate.

JDBC este o interfata de programare şi o infrastructura pentru accesul la o baza de date relationala. JDBC isi propune sa standardizeze mecanismul de conectare, de trimitere a cererilor, de lucru cu tranzactii, cat şi de reprezentare a rezultatelor unei cereri. Programatorul este insa liber a foloseasca ce dialect SQL doreste, in funcţie de serverul de baze de date cu care lucreaza.

JDBC propune o solutie multinivel:

- La nivelul cel mai de sus, o aplicatie Java executa cereri SQL prin intermediul unei interfete de programare definita in pachetul java.sql. Acest pchet nu conţine altceva decat definitiile unor interfete Java. Implementarile acestor interfete sunt furnizate separat pentru diversele sisteme de baze de date.
- Aplicatia acceseaza baza de date prin intermediul unui program special numit *JDBC driver manger*.
- Acest program gestioneaza programele specifice de control (engl.*driver*) pentru fiecare sistem de baze de date in parte. Un program de control implementeaza interfata java.sql.Driver.

Principalele interfete definite in pachetul java.sql sunt:

- Driver. Acesta este interfata asociata cu programul de control. Ficare program de control

va furniza o clasa care implementeaza aceasta interfata. O astfel de clasa pentru MySQL este com.mysql.jdbc.Driver, iar pentru o punte JDBC-ODBC este sun.jdbc.odbc.JdbcOdbcDriver.

- DriverManager. Aceasta clasa gestioneaza programele de control disponibile pentru conectarea la diverse baze de date.
- Statement. Este o clasa utila pentru crearea şi executarea de cereri SQL.
- PreparedStatement. Este o subclasa a clasei Statement utila pentru crearea de enunturi SQL parametrizate şi care sa poata fi executate eficient.
- CallableStatement. Este o subclasa a clasei Statement utila pentru executarea de proceduri stocate.
- ResultSet. Se utilizeaza pentru reprezentarea multimii de rezultate obţinute in urma executiei unei interogari SQL.
- ResultSetMetaData şi DatabaseMetaData. Sunt clase pentru reprezentarea metadatelor referitoare la o multime de rezultate respectiv la o baza de date. Exemple de metadate sunt: numele atributelor unei relatii, tipurile atributelor unei relatii, diverse caracteristici ale sistemului de baze de date (standardul SQL, dacă suporta sau nu proceduri stocate, etc.)

Pasii de prelucrare la folosirea JDBC sunt urmatorii:

i) Incarcarea programului de control specific serverului de baze de date. Clasa incarcata conţine membrii statici ce creaza o instanta a programului de control şi il inregistreaza la gestionarul programelor de control.

```
Class.forName(com.mysql.jdbc.Driver);
```

ii) Definirea URL-ului conexiunii.

```
String dbUrl = "jdbc:mysql://127.0.0.1/magazin";
String utilizator = "";
String parola = "";
```

iii) Stabilirea conexiunii.

```
Connection c = DriverManager.getConnection(dbUrl, utilizator,parola);
```

iv) Crearea unui obiect pentru reprezentarea unei cereri SQL.

```
Statement s = c.createStatement();
```

v) Executarea cererii SQL.

```
ResultSet r = s.executeQuery("select isbn,prim_autor,titlu from Carte;");
```

vi) Prelucrarea rezultatelor.

```
while(r.next()) {
   System.out.println(
   r.getString(“isbn") + ", "
   + r.getString("prim_autor")
   + ": " + r.getString(“titlu") );
}
```

vii) Inchiderea conexiunii.

```
c.close();
```

Pentru tratarea erorilor in operatii SQL, JDBC furnizeaza clasa SQLException.

Se recomanda ca parametrii unei conexiuni sa fie indicati cu ajutorul unui fisier de proprietati:

```
driver=sun.jdbc.odbc.JdbcOdbcDriver
url=jdbc:odbc:carte
utilizator=admin
parola=admin
```

Dacă o anumita interogare SQL se executa de un numar repetat de ori, atunci este mai eficient sa se foloseasca un *enunt pregatit* (engl.*prepared statement*). In esenta un enunt pregatit este o interogare compilata şi optimizata pentru fi executata mai eficient.

Un enunt pregatit este in acelasi timp şi un *sablon de interogare*. Un astfel de sablon conţine niste parametri specificati prin semnul intrebarii. Valoarea acestor parametri se completeaza inaintea executiei interogarii.

```
String sablon= "update Angajat set salariu=? where id=?";
PreparedStatement cerere =
  conexiune.preparedStatement(sablon);
float[] salarii_noi = getSalariiNoi();
int[] id_angajati = getIdAngajati();
for (int i=0;i< id_angajati.length;i++) {
  cerere.setFloat(1,salarii_noi[i]);
  cerere.setInt(2,id_angajati[i]);
  cerere.execute();
}
```

Metadatele sunt date pentru descrierea altor date. JDBC conţine un numar de clase utile pentru extragerea de metadate referitoare la baze de date, multimi de raspunsuri, tabele, programe de control, etc.

Metadatele sunt utile in urmatoarele situatii:

- Pentru a testa dacă sistemul de baza de date are anumite facilitati, a caror lipsa ar putea produce erori. O astfel de facilitate este spre exemplu disponibilitatea procedurilor stocate.
- Dacă nu se cunoaste structura bazei de date.

Tipuri de metadate:

- Metadate referitoare la baza de date. Se obţin cu ajutorul clasei DatabaseMetaData. Aceste metadate includ: numele programului de control, versiunea programului de control, verificarea disponibilitatii standardului SQL2, verificarea disponibilitatii unor facilitati specifice ale SQL, etc.
- Metadate referitoare la multimea de rezultate. Se obţin cu ajutorul clasei ResultSet. Aceste metadate includ: numarul de atribute din multimea de rezultate, numele atributelor, numl relatiei din care s-au obţinut atributele, tipurile acestor atribute, etc.

Metadatele pot fi folosite pentru implementarea unor programe utilitare. Un exemplu ar putea fi un program utilitar pentru executarea unei interogari generale şi salvarea rezultatelor impreuna cu diverse informaţii referitoare la baza de date.

Intr-o aplicatie de comert electronic se inglobeaza obligatoriu o baza de date. De asemenea, o astfel de aplicatie va conţine şi un nivel de obiecte din omeniul problemei (engl.*business objects*). Se pune problema maparii acestor obiecte in relatii ale modelului relational. Pentru aceasta se creaza cate o clasa corespunzatoare ficarei relatii a bazei de date, cate o variabila membru pentru ficare atribut al acestei relatii şi metode corespunzatoare de acces la aceste variabile membru.

Codul metodelor de acces va folosi cereri SQL pentru acces la atributele corespunzatore din baza de date.

Deschiderea unei conexiuni la o baza de date este un proces consumator de timp. El poate dura mult mai mult decat executarea unei cereri. Din acest motiv se pune problema refolosirii conexiunilor in aplicatiile care se conecteaza de un numar repetat de ori la o baza de date. Se foloseste o clasa speciala numita *rezerva de conexiuni* (engl.*connection pool*). Aceasta clasa ruleaza pe un fir separat şi are urmatoarele responsabilitati:

- Prealocarea conexiunilor pe constructorul clasei. Constructorul mai primeste un numar maxim de conexiuni, un numar initial de conexiuni şi un indicator dacă rezerva sa astepte sau nu cand se cere o conexiune şi nu exista nici una libera.
- Gestiunea conexiunilor disponibile. Dacă se cere o conexiune şi exista o conexiune libera, atunci ea este alocata şi trecuta in lista conexiunilor ocupate.

- Alocarea de noi conexiuni. Dacă se cere o conexiune, nu exista nici una libera şi nu s-a atins numarul maxim de conexiuni, atunci se creaza o noua conexiune pe un fir separat. Dacă insa s-a atins limita maxima, se asteapta sau nu in funcţie de indicatorul setat in constructor.
- Asteptarea pentru eliberarea unei conexiuni.
- Inchiderea conexiunilor.

Intr-o aplicatie cu miniservere, rezerva de conexiuni poate fi alocata unui singur miniserver sau se poate partaja intre mai multe miniservere.

5.5. LIMBAJUL DE METAMARCARE XML

XML este un acronim pentru *eXtensible Markup Language – limbaj extensibil de marcare*. XML este un *limbaj de meta-marcare* - un limbaj pentru definirea limbajelor de marcare. *Limbaj de marcare* (engl.*markup language*) = un limbaj pentru descrirea formei unui document (cum trebuie interpretat conţinutul documentului). Un limbaj de marcare defineste un *tip de document*. Un document scris intr-un limbaj de marcare este compus din *date* şi *marcaje*. *Marcajele* sunt acele informaţii care indica utilizatorului cum trebuie interpretat conţinutul documentului (datele propriu-zise).

Exista diverse tipuri de marcaje:
- *Stilistice*: indica stilul de prezentare a documentului: I, U, B, FONT, etc.
- *Structurale*: indica cum este structurat documentul: HEAD, BODY, P, H, etc
- *Semantice*: indica semnificatia conţinutului documentului: TITLE, META, etc.

Un limbaj de marcare creat cu XML se numeste şi *aplicatie XML* (ex. XHTML).

Utilizarea XML conferă următoarele **avantaje:**
- Faciliteaza interoperabilitatea datelor intre aplicatii.
- Ofera o alternativa la raspandirea *formatelor proprietare*.
- Atat datele cat şi marcajele sunt stocate sub forma de text, fapt ce permite manevrarea şi editarea usoara, practic cu orice editor de texte.
- Fiind un limbaj de meta-marcare, este configurabil.
- Poate fi citit şi interpretat relativ usor de un agent uman.
- Este standardizat, ceea ce inseamna ca este relativ unanim cunoscut şi acceptat.
- Este liber in sensul ca specifcatia este disponibila liber, este independent de platforma şi este bine sustinut de unelte software disponbile liber.

Tipuri de aplicatii care folosesc XML sunt:
- Interactiunea cu doua sau mai multe surse de date, reprezentate in formate diferite = *integrarea datelor* (engl.*data integration*) disponibile de la aplicatii diferite putand rula pe platforme diferite
- Transferarea unei parti a prelucrarilor asupra datelor pe calculatoarele client in arhitecturile client-server
- Furnizarea de vederi diferite asupra acelorasi date
- Agenti pentru colectarea de informaţii

Documente XML bine-formate

Cerinta de a fi *bine-format* (engl.*well-formed*) este o constrangere de baza impusa documentelor XML. Un document *XML bine-format* trebuie sa respecte regulile de sintaxa incluse in specificatia XML 1.0, adica:

- Documentul conţine unul sau mai multe elemente. Un element - un nod al unui arbore.
- Exista un singur element - *radacina* - care conţine in interiorul sau celelalte elemente;
- Fiecare element, exceptand radacina, trebuie incuibat corect in cadrul unui element cuprinzator numit *parintele* sau.

Un document XML conţine *date* şi *marcaje*. Datele sunt reprezentate sub forma unor secvente de caractere (engl.*character data*). Marcajele indica structura documentului (sunt *metadate*). Ele includ:

- Marcajele de inceput şi sfarsit ale elementelor;
- Marcajele elementelor vide;
- Comentariile;
- Referintele la entitati;
- Delimitatorii sectiunilor CDATA;
- Definitiile de tip de document (engl.*Document Type Definition* – DTD);
- Instructiunile de prelucrare.

Un document XML se compune din trei parti:

- Declaratie; indica faptul ca avem un document XML.
- *Prolog* (*antet)* - poate fi vid. Conţine o definitie de tip sau o referinta la definitia de tip.
- *Elementul radacina*, care conţine in interiorul sau toate celelalte elemente.

Un exemplu îl constituie **arborii binari în XML:**

```
<?xml version="1.0" standalone="yes"
    encoding="UTF-8" ?>
<!– Un arbore binar reprezentat în XML -->
<ARBBIN>
  <INFO>a</INFO>
  <ARBBIN>
    <INFO>b</INFO>
    <ARBBIN/>
    <ARBBIN>
      <INFO>d</INFO>
      <ARBBIN/>
      <ARBBIN/>
    </ARBBIN>
  </ARBBIN>
  <ARBBIN>
    <INFO>c</INFO>
    <ARBBIN>
      <INFO>e</INFO>
      <ARBBIN/>
      <ARBBIN/>
    </ARBBIN>
    <ARBBIN/>
  </ARBBIN>
</ARBBIN>
```

Prima linie din programul prezentat este *declaratia* de document XML. Textul inclus între <!- şi --> este un *comentariu*. Elementul de baza pentru structurarea unui document este *elementul*. Un element este inclus între un marcaj de început şi un marcaj de sfarşit.

<ARBBIN> este un *marcaj de început* şi </ARBBIN> este un *marcaj de sfarşit*.

Primul marcaj <ARBBIN> începe *elementul radacina* al documentului.

<ARBBIN/> este un *marcaj vid* şi este echivalent cu <ARBBIN></ARBBIN>. Indica un *element vid*. Elementul <ARBBIN>a</ARBBIN> conţine textul ”a” în interiorul sau.

Elemente XML cu atribute:

- Informaţia din cadrul unui nod al arborelui se poate reprezenta ca valoare a unui *atribut*.

```
...
  <ARBBIN>
    <INFO info="b" />
    <ARBBIN/>
    <ARBBIN>
      <INFO info="d" />
        <ARBBIN/>
      <ARBBIN/>
    </ARBBIN>
  </ARBBIN>
  ...
```

- Un element poate avea mai mult decat un atribut.

```
<INFO cheie="10" info="b" />
```

Documente XML valide

Validitatea este o constrangere aditionala care poate fi impusa documentelor XML. Ea se bazeaza pe ideea de *tip de document* mostenita din SGML.

Un document XML se numeste *valid* dacă este bine format şi dacă exista o definitie de tip de document – DTD cu care documentul este consistent. Se spune ca documentul *este o instanta* a acestei definitii de tip.

Adaugand o DTD documentului XML care conţine un arbore binar se obţine:

```
<?xml version="1.0" standalone="yes" ?>
<!– Un arbore binar reprezentat în XML -->
<!DOCTYPE ARBBIN [
  <!ELEMENT ARBBIN (INFO,ARBBIN,ARBBIN)? >
  <!ELEMENT INFO (#PCDATA) >
]>
<ARBBIN> ...
```

În acest caz, DTD-ul este conţinut în cadrul aceluiasi fisier cu documentul XML. DTD-ul este inclus între marcajele <!DOCTYPE ARBBIN [ ]>. Marcajul de start al DTD-ului indica elementul radacina al documentului.

DTD-ul conţine o declaratie pentru fiecare element al documentului.

Fiecare declaratie de element specifica numele elementului şi un *model al conţinutului* sau. DTD-ul poate fi plasat într-un fisier separat. Dacă presupunem ca DTD-ul este plasat în fisierul *arbbin.dtd*, atunci va trebui sa schimbam doar specificatia DTD-ului din documentul XML:

```
<!DOCTYPE ARBBIN SYSTEM "arbbin.dtd">
```

Fisierul *arbbin.dtd* va conţine doar declaratiile elementelor:

```
<!ELEMENT ARBBIN (INFO,ARBBIN,ARBBIN)? >
<!ELEMENT INFO (#PCDATA)>
```

Dacă informaţia dintr-un nod se reprezinta ca valoare a atributului *info* al elementului INFO atunci DTD-ul este:

```
<!ELEMENT ARBBIN (INFO,ARBBIN,ARBBIN)? >
```

```
<!ELEMENT INFO EMPTY>
<!ATTLIST INFO
  info CDATA #REQUIRED>
```

Definitii de tip de document

O definitie de tip de document specifica:

- Elementul radacina al documentului
- Numele, atributele şi modelul de conţinut al fiecarui element din document:

<!ELEMENT *nume_element model_de_conţinut* >

Modelul de conţinut al unui element se specifica cu !ELEMENT şi poate fi:

- *Vid, specificat prin EMPTY*
- *Text, specificat prin #PCDATA*
- *Alte elemente structurate în secventa (,), optionale (?), alternative (|) sau repetitive (* şi +)*

Atributele se specifica pentru fiecare element în parte cu !ATTLIST incluzand numele atributului, tipul valorii sale şi setarea valorii implicite:

<!ATTLIST *nume_element*
nume_atribut tip_atribut setare_valoare_implicita
...
nume_atribut tip_atribut setare_valoare_implicita >

Tipuri de atribute pot fi: enumerare, CDATA, etc. Setari de valoare implicita pot fi: #IMPLIED, #REQUIRED, #FIXED, etc.

Entități, șiruri de caractere neanalizate, spatii de nume

Orice procesor de XML trebuie sa distinga între date şi marcaje. Pentru a putea desemna caracterele < > ' şi " ca date trebuie sa folosim *entitatile predefinite* < > ' şi ". Pentru & se foloseste entitatea predefinita &.

Caracterele speciale UNICODE pot fi referite &#*codZecimal*; sau &#x*codHexazecimal*;. Astfel © se specifica prin © sau ©.

Pentru a indica faptul ca un sir de caractere trebuie sa nu fie analizat de procesorul documentului XML se foloseste o sectiune CDATA:

```
<INFO><![CDATA[Text într-un element <INFO>]]></INFO>
```

Entitatile parametrice sunt asemanatoare cu macroinstructiunile şi sunt utile pentru organizarea unui DTD.

```
<!ENTITY % nume "lista_nume">
```

Numele de elemente şi atribute dintr-un document XML pot defini un *vocabular partajat* destinat reutilizarii în cadrul unor aplicatii multiple. Pentru evitarea coliziunilor de nume între astfel de vocabulare partajate se folosesc spatiile de nume (engl.*namespaces*).

Limbaje formale şi analizoare sintactice

XML este un metalimbaj formal pentru definirea de limbaje formale specifice. Limbaj formal = o multime de secvente de simboluri (enunturi, fraze) construite pe baza unei multimi de reguli de formare numita *sintaxa* limbajului şi interpretate potrivit unui anumit înteles numit *semantica* limbajului.

Pentru analiza şi prelucrarea enunturilor unui limbaj formal (a documentelor XML în

particular) este necesar un program special numit *procesor de limbaj*. O componenta de baza a unui procesor de limbaj este *analizorul sintactic* (engl.*parser*). Un analizor sintactic pentru un limbaj formal *L* primeste la intrare o multime de enunturi şi îndeplineste urmatoarele funcţii:

- Decide dacă enunturile verifica sintaxa limbajului *L*.
- Dacă nu, indica cat mai precis posibil ce reguli sunt încalcate şi unde anume.
- Dacă da, produce o reprezentare structurata intermediara a enunturilor astfel încat ele sa poata fi ulterior interpretate şi prelucrate conform semanticii lui *L*.

Un programator de aplicatii va fi interesat în primul rand cum poate integra un procesor de limbaj în cadrul unei aplicatii. În cazul XML el se va confrunta cu problema integrarii analizoarelor sintactice de XML. În general exista doua modalitati prin care un analizor sintactic poate comunica rezultatele analizei celorlalte componente ale aplicatiei: i) prin evenimente – *analizor sintactic bazat pe evenimente* (engl.*event-driven parser*); ii) prin construirea explicita a unei structuri intermediare care sa descrie complet enunturile analizate – *translator*.

Analiza şi prelucrarea documentelor XML folosind Java se face astfel:

- Un analizor sintactic de XML poate fi i) *cu validare* (engl.*validating parser*), caz în care el verifica consistenta documentului XML cu un DTD sau ii) *fara validare* (engl.non-*validating parser*), caz în care el verifica doar dacă documentul XML este bine-format.
- Exista o standardizare a interfetelor de programare (engl.*Application Programming Interface* – API) pe care trebuie sa le implementeze un analizor sintactic de XML astfel încat el sa poata fi integrat cat mai usor şi portabil în cadrul unei aplicatii. În prezent exista doua API standard:
  - Simple API for XML – SAX, un standard *de-facto* existent în doua versiuni SAX1 şi SAX2. Defineste o multime de interfete pentru un analizor sintactic bazat pe evenimente (engl.*event-driven parser*). SAX a fost definita întai pentru Java, dar s-au definit ulterior versiuni similare şi pentru alte limbaje (C++);
  - Document Object Model – DOM, un standard W3C pentru un analizor sintactic care traduce un document XML într-o structura de date abstracta de tip arbore numita DOM. DOM se doreste a fi o specificatie independenta de limbaj, fiind valabila şi pentru limbajele de scripting (JavaScript)
- Se foloseşte pachetul XML for Java de la IBM (*XML4J*). El se bazeaza pe proiectul *xml.apache.org* din cadrul Apache Software Foundation, în speta pachetul *Xerces*.

Standardul SAX

SAX este un standard *de facto* pentru o interfata de programare a unui mecanism de analiza sintactica bazat pe evenimente a documentelor XML. Un programator de aplicatii va putea folosi orice biblioteca de programe de analiza sintactica conforma cu standardul SAX, fiind suficient sa cunoasca doar aceasta interfata standard.

Termenul *bazat pe evenimente* se refera la folosirea unor funcţii *call-back*. Aceste funcţii trebuie furnizate de programatorul de aplicatii şi ele trateaza o serie de evenimente de interes primite de la analizorul sintactic. Un analizor sintactic conform cu SAX va trebui sa implementeze interfata SAX. Funcţiile *call-back* sunt implementate de obiecte speciale de tratare a evenimentelor (engl.*handler)*. Un astfel de obiect trebuie sa implementeze niste interfete standard. O biblioteca conforma cu SAX va trebui sa conţina partea de interfete standard SAX şi o implementare particulara a acestor interfete. O astfel de bibliotca este *Xerces*.

O interfaţă SAX conţine trei parti:

- Partea de baza org.xml.sax. Aici se definesc interfetele standard ale obiectelor de tratare a evenimentelor SAX.
  - *ContentHandler – evenimente referitoare la conţinut*
  - *DTDHandler – evenimente referitoare la DTD*

- *ErrorHandler – evenimente de eroare*
- *EntityResolver – evenimente de tratare a entitatilor*
- *XMLReader – interfata unui analizor sintactic XML orientat pe evenimente*
- *Attributes – interfata pentru o lista de atribute XML*

– Partea de clase de ajutor org.xml.sax.helpers. Aici se definesc niste implementari implicite pentru interfetele SAX.

- *DefaultHandler – implementare implicita a tuturor celor 4 interfete pentru tratarea evenimentelor SAX.*
- *AttributesImpl – implementare implicita pt o lista de atribute XML*
- **...**

– Partea de extensii org.xml.sax.ext.

Folosirea SAX presupune parcurgerea urmatorilor pasi:

– Furnizarea de implementari pentru obiectele de tratare a evenimentelor SAX. O metoda comoda este extinderea implementarii implicite din clasa DefaultHandler. Aceasta clasa furnizeaza implementari implicite pentru toate cele 4 interfete standard SAX. În cadrul acestei implmentari se vor redefini doar metodele care sunt necesare.
– Se creeaza o clasa pentru reprezentarea unui analizor sintactic. Pentru aceasta este util sa se utilizeze o biblioteca care sa dispuna deja de o astfel de implementare. Spre exemplu, pachetul *Xerces* dispune de clasa SAXParser.
– În programul principal se realizeaza urmatoarele prelucrari:
  – *Se creaza un obiect de tratare a evenimentelor SAX..*
  – *Se creaza un obiect pentru reprezentarea unui analizor sintactic.*
  – *Se înregistreaza obiectul de tratare a evenimentelor la analizorul sintactic.*
  – *Se executa funcţia de analiza sintactica a obiectului analizor.*
– În funcţiile de tratare a evenimentelor de analiza sintactica se realizeaza toate prelucrarile necesare din cadrul aplicatiei respective. Spre exemplu, se pot folosi aceste evenimente pentru extragerea informaţiilor relevante pentru aplicatie din cadrul documentului XML şi reprezentarea lor sub forma unei structuri intermediare. Aceasta structura se poate prelucra ulterior.

Interfaţa DOM

DOM (nivelul 2) este o platforma şi o interfata independenta de limbaj ce permite programelor şi scripturilor sa acceseze şi sa actualizeze dinamic conţinutul şi structura documentelor XML.

Pachetul de baza org.w3c.dom poate fi gandit ca o specificatie a unui tip abstract de date pentru reprezentarea unui document XML sub forma unui arbore. Principalele interfete definite în acest pachet sunt urmatoarele:

- Node – este tipul primar al unui nod dintr-un arbore DOM
- NodeList – reprezinta o colectie ordonata de noduri
- Element – reprezinta un element al unui document XML Extinde tipul Node.
- NamedNodeMap – reprezinta o colectie de noduri ce pot fi accesate prin nume (de exemplu nodurile atribut unui element XML).
- Document – reprezinta nodul document al unui arbore DOM. Extinde tipul Node.
- Attr – reprezinta un nod atribut al unui element XML. Extinde tipul Node.
- Text –
- Comment –
- Entity –
- ProcessingInstruction –
- **...**

La fel ca şi pentru SAX, folosirea DOM presupune existenta unei biblioteci care sa implementeze interfetele DOM. O astfel de biblioteca este pachetul *Xerces*.

Folosirea DOM presupune parcurgerea urmatorilor pasi:

- Implementarea unui modul de prelucrare a informaţiei dintr-un document XML reprezentat sub forma unui arbore DOM.
- În programul principal se realizeaza urmatoarele prelucrari:
  - *Construirea unui obiect pentru reprezentarea unui analizor sintactic conform cu DOM. Pachetul* Xerces *furnizeaza pentru aceasta clasa DOMParser.*
  - *Se executa funcţia de analiza sintactica a obiectului analizor.*
  - *Se extrage obiectul document şi se prelucreaza.*

```
DOMParser parser = new DOMParser();
parser.parse( ... );
Document doc = parser.getDocument();
Element root = doc.getDocumentElement();
...
```

Limbajele XSL şi XSLT

XSL (Extensible Stylesheet Language) este un limbaj de prelucrare a documentelor XML. XSL este o aplicatie XML.

XSL are doua componente relativ distincte:

- *Limbajul de transformare* XSLT. XSLT permite trasformarea unui document XML într-un alt document XML sau document text. XSLT 1.0 este o recomandare W3C din 16 Noiembrie 1999. XSLT foloseste limbajul XPath pentru adresarea diverselor parti din arborele unui document XML.
- *Limbajul de formatare* XML-FO. XML-FO permite descrierea formatului de prezentare a unui document XML prin furnizarea unui vocabular de formatare.

Pentru reprezentarea unui document XML sunt necesari în general doi pasi:

- *Transformarea* documentului XML initial într-un document XML care foloseste vocabularul XML-FO. Acest pas poate optional sa realizeze şi o filtrare a conţinutului documentului initial.
- *Formatarea* documentului XML-FO rezultat, cu ajutorul unui program special de formatare, adica transformarea sa într-un format specific de prezentare/ afisare/ vizualizare. Exemple de astfel de formate sunt PDF, Postscript şi RTF.

În WWW se poate folosi XSLT pentru a transforma un document XML într-un document XHTML sau WML. Transformarea poate avea loc fie la server, fie la client.

Transformari XSL

XSLT este un *limbaj de transformare* a unui document XML într-un alt format. Formatul de iesire poate fi tot XML (de exemplu XHTML sau WML) sau alt format text.

Pentru producerea unei transformari este nevoie de trei elemente:

- *Sursa transformarii*, documentul XML de intrare
- *Specificatia transformarii*, un document XSLT
- *Motorul transformarii*, un program numit *procesor XSLT*. Acest program aplica specificatia XSLT a transformarii, sursei transformarii.

Concepte de baza XSLT:

- Documentul XML furnizat ca intrare a unei transformari XSLT este de forma unui arbore.
- XSLT modeleaza un document sub forma de arbore, similar cu XPath. XPath este un limbaj de expresii pentru adresarea unei parti dintr-un document XML.
- XPath şi XSLT recunosc 7 tipuri de noduri ale arborelui unui document XML de intrare: i)

Radacina documentului: este nodul de start al oricarui document; ii) *Element*: reprezinta un element al sursei XML. iii) *Text*: reprezinta un conţinut text dintr-un element al sursei XML. iv) *Atribut*: reprezinta un atribut al unui element din sursa XML; v) *Spatiu de nume*: vi) *Comentariu*: vii) *Instructiune de prelucrare*: reprezinta textul unei instructiuni de prelucrare din sursa XML.

- Pe multimea nodurilor unui document este definita o relatie de ordonare liniara numita *ordinea documentului*. In esenta aceasta ordine exprima urmatoarele: i) nodul radacina este primul; ii) nodurile element apar înaintea copiilor lor; iii) atributele şi spatiile de nume asociate unui element apar înaintea copiilor elementului respectiv; iv) spatiile de nume asociate unui element apar înaintea atributelor elementului respectiv.v) ordinea relativa a spatiilor de nume şi a atributelor unui element este dependenta de implementare.
- Pentru fiecare nod se defineste o *valoare* a nodului. De exemplu, pentru nodul radacina valoarea se obţine concatenand conţinutul tuturor nodurilor descendente de tip text.

Modelul de prelucrare folosit de XSLT este următorul:

- O transformare XSLT descrie o multime de reguli pentru transformarea unui arbore de intrare într-un arbore de iesire. O *regula* conţine o *parte de conditii*, numita *sablon* (engl.*pattern*) şi un *tipar* (engl.*template*). Sablonul este identificat cu nodurile arborelui sursa. În cazul unei *identificari* (engl.*matching*), tiparul este instantiat şi astfel el genereaza o parte din arborele documentului de iesire.
- Prelucrarea unui arbore sursa se bazeaza pe parcurgerea recursiva a arborilor:
  - Prelucrarea de baza este aplicarea regulilor unei multimi de noduri din arborele sursa. Procesul începe prin aplicarea regulilor listei de noduri formata doar din nodul radacina.
  - Se obţine un arbore de iesire prin aplicarea regulilor unui singur nod din arborele sursa. În cazul aplicarii regulilor unei liste de noduri, portiunea care rezulta din arborele de iesire se obţine prin concatenarea arborilor obţinuti prin aplicarea regulilor fiecarui nod din lista.
  - Aplicarea regulilor unui nod presupune determinarea nodurilor din subarborele sau carora li se poate aplica cel putin o regula, potrivit ordinii documentului. Dacă pentru nodul selectat se pot aplica mai multe reguli, pentru a determina regula aplicabila se foloseste o strategie de rezolvare a conflictelor bazata pe specificitate. Regula cu partea de conditii este cea mai specifica va fi selectata. Nodul caruia i se aplica regulile devine *nodul curent*, iar lista de noduri din care a facut parte cand fost selectat este *lista nodurilor curente*.
  - Un tipar poate conţine instructiuni pentru selectarea altor noduri carora li se vor aplica regulile de transformare. Dacă lista acestor noduri este nevida, atunci procesul de aplicare a regulilor conţinua recursiv pentru aceasta lista pana cand nu se mai selecteaza noduri noi pentru prelucrare.

Şabloane în XSL

Un document XSLT conţine *reguli de transformare*. O regula de transformare are sintaxa urmatoare:

```
<xsl:template match="sablon">
  tipar
</xsl:template>
```

Specificarea sablonului se face cu atributul match şi se aplică următoarele reguli:

- Identificarea nodului radacina al documentului se face cu sablonul /.
- Identificarea unui element se face cu un sablon identic cu numele elementului.
- Identificarea unui element arbitrar se face cu sablonul *.
- Identificarea elementelor copil ale unui anumit element se face utilizand un sablon de forma *parinte*/*copil*. Mai general, se poate folosi un şablon de forma $e_1/e_2/\ldots/e_n$. El identifica elementele e_n care au un lant de stramosi de forma e_1, e_2, …, e_{n-1}. Oricare dintre

e_i poate fi *. Un șablon de forma /e_1/e_2/…/e_n identifica un lant de stramosi ce incepe din radacina.

- Identificarea elementelor descendent ale unui anumit element se face cu un sablon de forma *stramos*//*descendent*.
- Identificarea unui atribut se face cu un șablon de forma *@atribut*. Identificarea unui nod text, comentariu sau instructiune de prelucrare se face cu sabloane de forma text(), comment() respectiv processing_instruction().
- Identificarea după unul din mai multe sabloane posibile se face cu un sablon de forma s_1 | s_2 | ... | s_n.

Șabloane cu teste:

- Pe un nod se pot face teste suplimentare adăugând la un șablon un test de forma [*conditie*].
  - Identificarea unui element cu un atribut se face cu un șablon de forma: *element*[*@atribut*].
  - Identificarea unui element cu un atribut cu o anumită valoare se face cu un șablon de forma *element*[*@atribut*="*valoare*"].
  - Identificarea unui element *e* prin poziția sa în lista nodurilor frate care sunt elemente *e* se face cu un șablon de forma *element*[*pozitie*] sau *element*[position()=*pozitie*].
  - Testele se pot încuiba. De exemplu *element*[*copil*[*@atribut*]] identifică un element care are un element copil cu un anumit atribut.
  - Într-un test se poate folosi şi operatorul not. De exemplu, șablonul *e*[not *last*()] identifica toate elementele *e* care nu sunt ultimele în lista nodurilor frate care sunt elemente *e*.
  - Testele pe un nod se pot combina cu caile de noduri pentru a construi șabloane complexe. De exemplu, *e*[*@a*="*valoare*"]//e_2 identifică elementele e_2 care sunt descendente ale unor elemente e_1 cu atributul *a* cu valoarea *v*.

Un exemplu de transformare trivială ce poate fi aplicată oricărui document XML, cu același rezultat, generarea textului Ceva banal, este următorul:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform
  version="1.0">
  <xsl:template match="/">
    Ceva banal.
  </xsl:template>
</xsl:stylesheet>
```

xsl:value-of determina valoarea unui nod. Nodul se specifica cu atributul select, relativ la nodul curent.

xsl:apply-templates determina aplicarea recursiva a regulilor de transformare unor noduri selectate cu atributul select, relativ la nodul curent identificat.

Cel mai simplu exemplu este regula implicita pentru nodul radacina, respectiv pentru fiecare element din documentul sursa:

```
<xsl:template match="/ | *">
        <xsl:apply-templates/>
</xsl:template>
```

Această regulă are ca efect aplicarea regulilor copiilor nodului radacina şi copiilor tuturor nodurilor element.

xsl:value-of se va folosi direct doar în contexte în care este evident nodul a carui valoare va fi selectata. În cazul în care pot fi selectate mai multe noduri, numai valoarea primului dintre ele va fi generata.

În cazul în care trebuie generate valorile nodurilor selectate, exista doua solutii:

- Generarea unui apel recursiv folosind xsl:apply-templates.

```
<xsl:template match="parinte">
```

```
  <xsl:apply-templates select="copil"/>
</xsl:template>
<xsl:template match="copil">
  <xsl:value-of select="."/>
</xsl:template>
```

– Folosirea xsl:for-each realizand astfel prelucrarea explicita a fiecarui nod selectat.

```
<xsl:template match="parinte">
  <xsl:for-each select="copil">
    <xsl:value-of select="."/>
  </xsl:for-each>
</xsl:template>
```

Expresii XPath

xsl:apply-templates, xsl:value-of şi xsl:for-each folosesc atributul select pentru selectarea unei multimi noduri ce fac obiectul unor prelucrari. Atributul select apare de asemenea şi în xsl:copy-of, xsl:variable, xsl:param şi xsl:sort.

Valoarea atributului select este o expresie XPath. Şabloanele de identificare a nodurilor sunt o submultime a multimii expresiilor XPath.

O expresie XPath este de unul din urmatoarele 4 tipuri: multime de noduri, logic, numar, sir de caractere.

O expresie care selectează o multime de noduri se numeste *expresie cale* (engl.*location path*). O cale este compusa dintr-un numar de *pasi* (engl.*location step*). Un pas are un *nod context*. Fiecare subcale selecteaza o multime de noduri relativ la un nod context. Fiecare nod din aceasta multime este nod context pentru pasul urmator.

Un pas se compune din 3 componente:

– O *axa* ce specifica relatia dintre nodurile selectate de pas şi nodul context.
– Un *identificator de nod* care specifica tipul şi numele nodurilor selectate
– Zero sau mai multe predicate care pot filtra suplimentar multimea de noduri selectate.

BIBLIOGRAFIE

Baze de date

1. Codd, E.F., ***The Relational Model for Database Management***, Addison-Wesley, 1990
2. Date, C.J., ***An Introduction to Database Systems***, Addison-Wesley, Boston, 2004
3. Dolliner, R., ***Baze de Date şi Gestiunea Tranzacţiilor***, Editura Albastră, Cluj, 1997
4. Forta, B., ***SQL pentru Începători***, (SAMS) Teora, 2002 (traducere Daniel Cihodaru, după ediţia din 1999)
5. Fotache, M., Proiectarea ***bazelor de date. Normalizare şi postnormalizare. Implementări SQL şi Oracle***, Ed. Polirom, Iaşi, 2005
6. Ionescu, F., ***Baze de Date Relaţionale şi Aplicaţii***, Editura Tehnică, Bucureşti, 2004
7. Oppel, A., ***SQL fără mistere***, Ed. Rosetti Educational, Bucureşti, 2006
8. Peterson, J., ***Baze de date pentru începători***, Ed. All, 2003
9. Welling, L., Thomson, L., ***Dezvoltarea aplicaţiilor cu PHP şi MySQL***, Teora, Bucureşti, 2005
10. *** ***Sistemul de gestiune SQL Server***, http://www.microsoft.com/sql
11. *** ***Sistemul de gestiune MySQL*** , http://www.mysql.com

Programare orientată obiect și structuri de date

12. C. Bologa, ***Algoritmi și structuri de date***, Editura RISOPRINT, Cluj-Napoca, 2005
13. A. Carabineanu, ***Structuri de date***, Editura Matrixrom, 2006
14. V. Creţu, ***Structuri de date şi algoritmi***, Editura Orizonturi universitare, Timişoara, 2000
15. J. Knuth, ***Arta programării calculatoarelor***, Editura Teora, 1999
16. I. Ignat, C.L.Ignat, ***Structuri de date si algoritmi*** , ed. Albastra, Cluj, 2007
17. I. Ivan, M. Popa, P. Pocatilu, (coordonatori), ***Structuri de date. Tipologii de structuri de date***, Editura ASE, 2009
18. G. Soava, ***Programare orientată obiect, Teorie și aplicații***, Ed. Universitaria, Craiova, 2010
19. F. Wempen, ***Visual Studio 9***, Editura Teora, Bucureşti, 2007

Sisteme de operare

20. Acostăchioaie D., ***Administrarea şi configurarea sistemelor Linux***, ediţia a II-a, Ed. Polirom, Iaşi, 2003
21. Acostăchioaie D., Buraga S.,***Utilizare Linux. Noţiuni de bază şi practică***, Ed. Polirom, Iaşi, 2004
22. Cristea V., ş.a., ***UNIX***, Ed. Teora, Bucureşti, 1993
23. Ignat I., ş.a., ***Sistemul de operare UNIX. Gestionarea fişierelor***., Ed. MicroInformatica, Cluj-Napoca, 1992
24. Ignat I., Kacso A. ***, UNIX – Gestiunea proceselor***, Ed. Albastră, Cluj-Napoca, 1995
25. Pilat F., ***UNIX***, Ed. Teora, Bucureşti, 1992
26. Tanenbaum A. S., ***Modern Operating Systems***, Prentice Hall, 2001
27. Wielsch M., ***Le grand livre de UNIX***, Editions MicroApplication, Paris, 1994
28. *****, http://www.redhat.com
29. *****, http://metalab.unc.edu/pub/Linux/docs

Proiectarea sistemelor informatice

30. Zaharie, D., Rosca, I., ***Proiectarea obiectuala a sistemelor informatice,*** Dual Tech, Bucuresti, 2002
31. Oprea, D., ***Analiza si proiectarea sistemelor informationale economice,*** Editura Polirom, Bucuresti, 1999

32. Lungu, I., Sabau, Gh. s.a., ***Sisteme informatice - Analiza, proiectare si implementare***, Editura Economica, Bucuresti, 2003
33. Stanciu V., ***Proiectarea sistemelor informatice de gestiune***, Ed. Cison, Bucureşti 2000
34. C. J. Date, ***An Introduction to Database Systems***, 8th Edition, published by PearsonEducation, Inc. Adison Wesley Higher Education, 2004

Programare Internet

35. T. Anghel, ***Programarea in PHP. Ghid practic,*** Editura Polirom, 2005
36. F. M. Boian, R. F. Boian, ***Tehnologii fundamentale Java pentru aplicatii Web***, Editura Albastra, 2009
37. S.C.Buraga, ***Proiectarea siturilor Web. Design si functionalitate***, Editura Polirom, 2006
38. S.Buraga, ***Programarea in Web 2.0***, Editura Polirom, 2007
39. S. Buraga, ***Tehnologii XML***, Editura Polirom, 2006
40. S. C. Buraga , ***Tehnologii Web***, Editura MATRIX ROM, Bucureşti, 2001
41. Coulouris, G., Dollimore, J, Kindberg, T, ***Distributed Systems. Concepts and Design*** (3rd ed.), Pearson Education, 2001
42. L.Sângeorzan, C. L. Aldea, ***Tehnologii internet***, Ed. Univ. Transilvania, 2003
43. Ceri , Stefano et al, ***Designing Data-Intensive Web Applications,*** Morgan Kaufmann, 2002
44. S Tudor, V. Huțanu, ***Crearea si programarea paginilor WEB***, ed. L&S SOFT, 2006