SÎRBU FLORICA CRISTINA

SGBD ORACLE

PLOIEȘTI

2020

SGBD ORACLE/ Sîrbu Florica Cristina-2020, Ploiesti

Lucrare publicată online www.cevmpl.ro

ISBN- 978-973-0-32585-0

C U P R I N S

CUVÂNT ÎNAINTE

17 Ianuarie 2006 ORACLE România şi Ministerul Educaţiei şi Cercetării semnează acordul prin care Compania Oracle România îşi afirma angajamentul de a susţine mediul educaţional prin implementarea şi dezvoltarea unui program destinat elevilor şi profesorilor de liceu din România "Academia Oracle pentru licee". Obiectivul: „Completarea programei şcolare pentru clasa a XII-a cu modulul de „Tehnologii de baze de date -Database Design and Programming with SQL”.

Începând cu anul şcolar 2007-2008 elevii au beneficiat de cursul „Tehnologii de baze de date -Database Design and Programming with SQL” în programa şcolară. care propune studiul conceptelor generale de proiectare şi programare de baze de date, oferind elevilor posibilitatea să-şi dezvolte competenţe şi aptitudini solide în utilizarea noilor tehnologii.

Studiul bazelor de date în învăţământul preuniversitar creează premisele implementării în România a economiei bazate pe cunoaştere şi a societăţii informaţionale, iar Oracle este cel mai mediatizat nume din această zonă IT datorită excelenţei tehnologice a produselor firmei Oracle din Redwood Shores California.

Oracle Internet Academy a pus la dispoziţia profesorilor instructori un sistem complex format dintr-un curs şi o interfaţă de realizare a aplicaţiilor de baze de date Oracle. Pe site-ul https://academy.oracle.com/pages/student_area, un profesor instructor are acces prin intermediul numelui de user şi a parolei la interfaţa cursului format din două module:

1. Database Design - Semestrul 1
2. Programming with SQL-Semestrul 2

Lucrare este proiectata pe capitole astfel:

Capitolul I *Proiectarea bazelor de date* prezintă ierarhizat într-o structură sistemic – integratoare noţiunile, tehnicile, procedeele de proiectare a bazelor de date folosind metoda ERD-urilor

Capitolul II *Programarea bazelor de date* prezintă instrucţiunile limbajului SQL structurate pe categorii, prin exemple care vizează etapele realizării unui proiect.

CAPITOLUL I

PROIECTEREA BAZELOR DE DATE RELAŢIONALE

1.1. Date, informaţii, modele de date

Datele iniţiale, neprelucrate sunt numere, litere, imagini, sunete care provin din măsurători sau observaţii disparate şi necoordonate între ele, persoane, lucruri, fenomene care au o reprezentate internă numerică (digitală) într-un calculator

Informaţiile se obţin prin prelucrarea şi organizarea datelor şi se prezintă sub formă de rapoarte, statistici, diagrame pe baza cărora se pot emite judecăţi de valoare şi decizii. Exemplu: notele obţinute de elevi la testul de informatică sunt date numerice Media notelor la testul de informatică determină nivelul clasei. Nota elevului este o dată, media notelor este informaţia pe baza căruia se pot emite judecăţi de valoare.

Baza de date este o colecţie organizată, structurată de date, proiectată şi folosită în scopul modelării activităţii unui tip de organizaţie sau proces organizaţional. O bază de date este o colecţie de date operaţionale folosite de către aplicaţiile sistem. Datele operaţionale sunt distincte de datele de intrare de ieşire. Datele de intrare sunt informaţii introduse în sistem din lumea exterioară, de obicei prin terminale. Datele de ieşire se referă la rapoartele, statisticile, mesajele extrase din sistem.

1.2. Modele de date - clasificarea modelelor

Modelul de date este o abstractizare, o descriere formală a schemei bazei de date definind o colecţie integrată de concepte necesare descrierii datelor: structura datelor, legăturile dintre acestea, semantica lor, precum şi constrângerile impuse, o structură ce simbolizează toate caracteristicile entităţilor esenţiale ce prezintă interes pentru utilizator, o reprezentare şi o reflectare a lumii reale.

Modelul ierarhic de bază date Datele descrise de acest model sunt structurate în mod ierarhic, sunt organizate într-o structură arborescentă. Un singur tabel din această bază de date acţionează ca „o rădăcină” a arborelui în timp ce alte tabele se comportă ca ramuri. O relaţie, o legătură între două tabele este reprezentată de conceptul părinte-copil. Un tabel părinte poate fi asociat cu unul sau mai multe tabele copil, dar un tabel copil nu poate fi asociat decât unui tabel părinte. Asocierile posibile sunt 1-1 şi 1-M.

Modelul reţea de bază date Datele descrise de acest model sunt structurate sub formă de reţea. Structura modelului reţea este formată din noduri şi structurile set. Un nod reprezintă o colecţie de înregistrări, iar o structură set stabileşte şi reprezintă o relaţie de tip unul la mai mulţi (1-M). ceea ce înseamnă că o înregistrare poate fi corelată cu una sau mai multe înregistrări din nodul membru dar o înregistrare din nodul membru este corelată numai cu o singură înregistrare din nodul posesor.

Modelul relaţional de bază date a fost conceput pentru prima dată în 1969 de dr. Edgar Frank Codd, informatician american de origine engleză, cercetător de la San Jose Research Laboratories ce aparţineau firmei IBM. Model relaţional a fost prezentat în lucrare de referinţă: „A Relational Model of Data for Large Shared Databanks" (un model relaţional de date pentru bănci de date partajate de mari dimensiuni) şi este fundamentat pe două ramuri ale matematicii: teoria mulţimilor şi logica predicatelor de ordin întâi

Caracteristicile modelului relaţional sunt:

• simplitatea, modelul relaţional poate fi descris cu ajutorul unui număr mic de concepte care se referă la relaţii (structuri de date bidimensionale ce au proprietăţi speciale), rânduri (datele aflate în cadrul relaţiilor), coloane (câmpurile datelor din rândurile corespunzătoare) şi chei (mecanismul de identificare şi asociere a rândurilor aflate în unul sau mai multe tabele);

• suportul teoretic solid pe care se bazează şi anume teoria matematică a seturilor, ceea ce înseamnă că toate operaţiile sunt încheiate cu succes, iar rezultatele operaţiilor sunt predictibile.

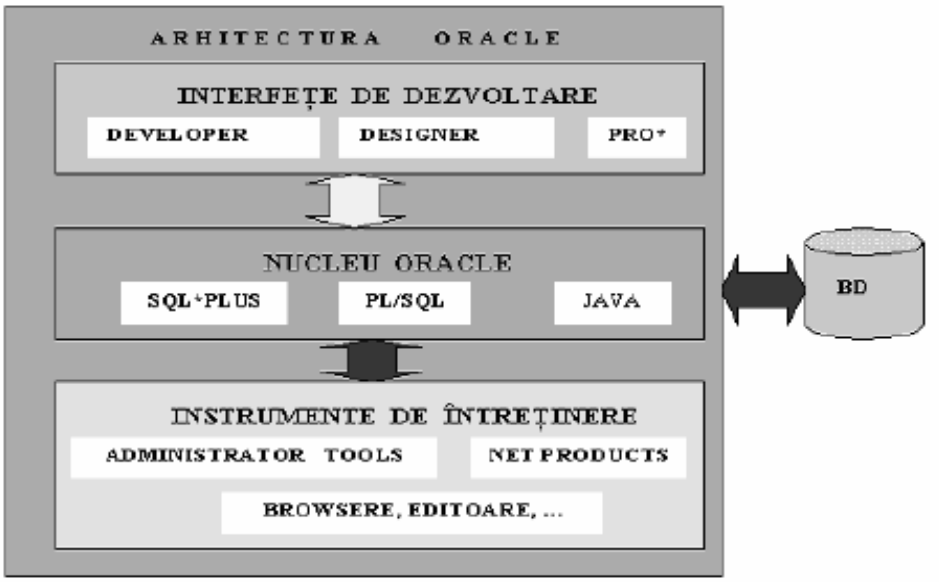
• modelul relaţional are trei componente:

1. componenta de structură a datelor;
2. componenta de manipulare a datelor
3. componenta de integritate a datelor (reguli necesare protecţiei datelor) .

Bază de date relaţională stochează datele în relaţii, pe care un utilizator le percepe ca tabele. Fiecare relaţie este compusă din înregistrări şi câmpuri, iar ordinea fizică a înregistrărilor sau a câmpurilor dintr-un tabel este complet lipsită de importanţă, fiecare înregistrare a tabelului este identificată, nu după locul unde se află, ci după un câmp care conţine o valoare unică şi nenulă. Acestea reprezintă cele două caracteristici ale bazei de dare relaţionale care permit datelor să existe independent de locul în care sunt stocate în calculator. În consecinţă, utilizatorul nu este obligat să cunoască locaţia fizică a unei înregistrări aşa cum se întâmplă la celelalte modele de bază de date (ierarhic şi reţea).

Modelul relaţional este studiat în această lucrare**.**

1.3. Sisteme de gestiune a bazelor de date relaţionale S.G.B.D –ul ORACLE



folosit pentru crearea, integrat unui SGBD (late.

Figura 1.1. Arhitectura ORACLE

SGBD-ul ORACLE realizat de firma Oracle Corporation U.S.A. este complet relaţional, robust, se bazează pe SQL standard extins. Componentele care formează arhitectura de bază Oracle sunt dispuse într-o configuraţie client/server care permite lucrul cu obiecte. Are BD Internet şi modul de optimizare a regăsirii datelor. Aceste componente sunt plasate pe calculatoare într-o reţea asigurând funcţionalităţi specifice, astfel: serverul asigură memorarea şi manipularea datelor, precum şi administrarea bazei de date iar clientul asigură interfaţa cu utilizatorul şi lansează aplicaţia care accesează datele din baza

Arhitectura Oracle este structurată pe trei niveluri: nucleul, interfeţele şi instrumentele de întreţinere. Nucleul conţine componentele care dau tipul relaţional pentru SGBD Oracle: limbajul relaţional de regăsire SQL şi limbajul procedural propriu PL/SQL.

Sistemul Oracle creează şi întreţine automat dicţionarul de date. Acesta face parte din baza de date Oracle şi conţine un set de tabele şi viziuni (vederi) accesibile utilizatorilor doar în consultare. Dicţionarul conţine informaţii de tipul: numele utilizatorilor autorizaţi, drepturile de acces, numele obiectelor din baza de date, structurile de date, spaţiul ocupat de date, chei de acces etc.

1.4. Etapele procesului de dezvoltare a bazelor de date relaţionale

1. Proiectarea modelului conceptual al bazei de date
2. Proiectarea modelului fizic al bazei de date
3. Proiectarea modelului operaţional al bazei de date

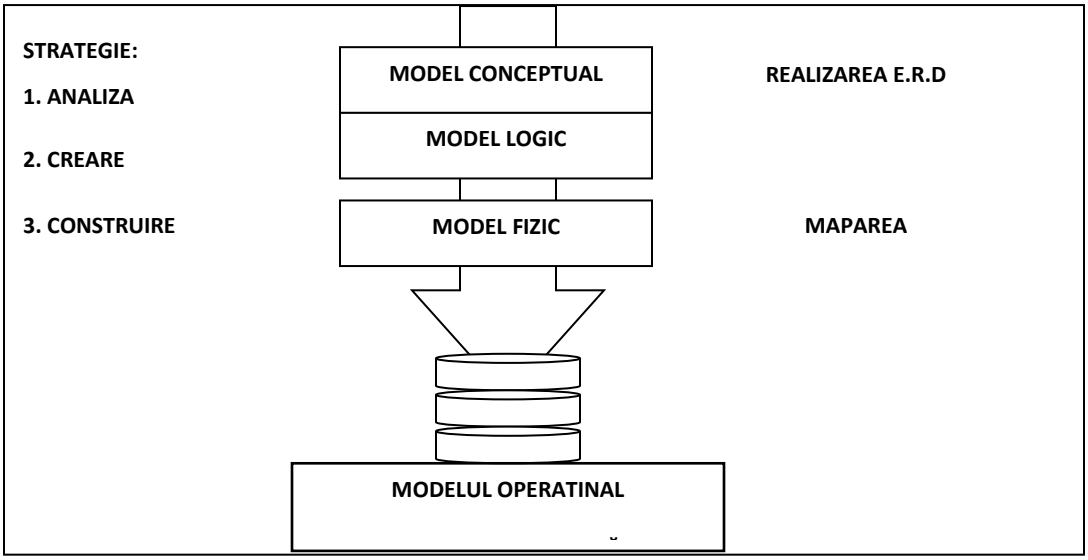


Figura 1.2. Etapele procesului de dezvoltare a bazei de date

Analiza datelor şi a modului de utilizare a acestora este etapa care implică examinarea atentă a activităţii modelate, pentru a evalua sistemul curent şi a analiza necesităţile viitoare, precum şi a estima necesităţilor informaţionale în vederea obţinerii unei imagini complete şi corecte asupra activităţii practice.

O analiză corectă are ca rezultat proiectarea unei baze de date care să corespundă obiectivelor propuse. Informaţiile necesare se obţin prin interviuri cu utilizatorii care interacţionează cu datele şi prin studierea documentelor folosite.

Crearea modelului conceptual al bazei de date (modelarea datelor şi a relaţiilor dintre ele) utilizează o metodă de proiectare simplă, grafică, cu diagrame entităţi, relaţie.

Diagrama entitate relaţie (ERD) este o reprezentare grafică, prin simboluri grafice, convenţionale, standardizate a fenomenelor, evenimentelor şi relaţiilor dintre ele.

Construirea este etapa de transformare modelului conceptual în model fizic, maparea, prin care se obţine o reprezentare a structurii tabelelor bazei de date.

Pentru implementarea modelului fizic în vederea obţinerii modelului operaţional se foloseşte SGBD-ului Oracle.

1.5. Proiectarea modelului conceptual Diagrama entităţi relaţie (ERD)

ERD-ul este modelul conceptual al bazei de date care integrează reprezentarea structurii şi a caracteristicilor bazei de date, prin două elemente grafice: entitatea şi relaţia.

1.5.1. Entităţi, Instanţe, Atribute

Entitatea este un substantiv la singular, tangibil (un obiect, persoană, un lucru, pacient, profesor, clădire), intangibil (nivel de pregătire, profil, categorie), eveniment (un concert, un fenomen, un proiect, o asociere,o conferinţă, operaţie)

Simbolul grafic de reprezentare a unei entităţi este dreptunghiul cu colţuri rotunjite care are scris, în interior , cu majuscule, substantivul care reprezintă entitatea

Entitatea este o clasă formată din mai multe obiecte.

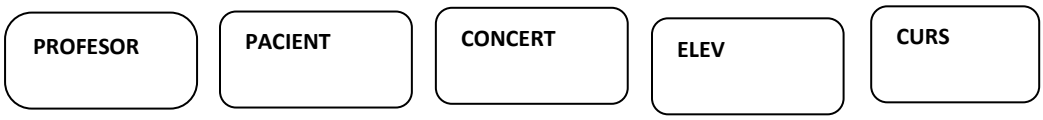


Figura 1.3. Exemple de entităţi şi modul de reprezentare

Instanţa este un obiect dintr-o entitate. Exemplu: entitatea PROFESOR este formată din profesorii liceului. Profesorul Mihaela Stanciu este o instanţă a acestei entităţi.

Atributul este o informaţie, un detaliu care identifică, clasifică, descrie, cuantifică sau exprimă starea unei instanţe dintr-o entitate. Atributele sunt substantive, la singular, scrise cu litere mici sub forma unei liste, în dreptunghiul cu colţuri rotunjite imediat sub numele entităţii. Atributele entităţii PROFESOR sunt: cnp, numele, prenumele, data_naşterii, telefon, adresa, disciplina, etc.

Caracteristica atributului este opţionalitatea introducerii în baza de date a valorii atributului Aceasta este reprezentată grafic prin caracterele: * ,# , o.

• Caracterul * (steluţa) indică un atribut obligatoriu care trebuie să aibă o valoare pentru fiecare instanţă a entităţii. (Este obligatoriu să introducem numele profesorilor)

• Caracterul o (cerculeţ) indică un atribut opţional pentru care nu este obligatorie introducerea unei valori pentru toate instanţele (telefon, e_mail)

• Caracterul # (diez) – identificator unic (UI**D)** este un atribut obligatoriu care identifică în mod unic o instanţă a entităţii; valoare introdusă trebuie să fie unică şi ne nulă (nr_matricol, cnp). Atributele identificatori unici sunt: identificator unic simplu (format dintr-un singur atribut) şi identificator unic compus format dintr-o combinaţie de atribute sau/şi o relaţie

Exemple de reprezentare a entităţilor şi atributelor:

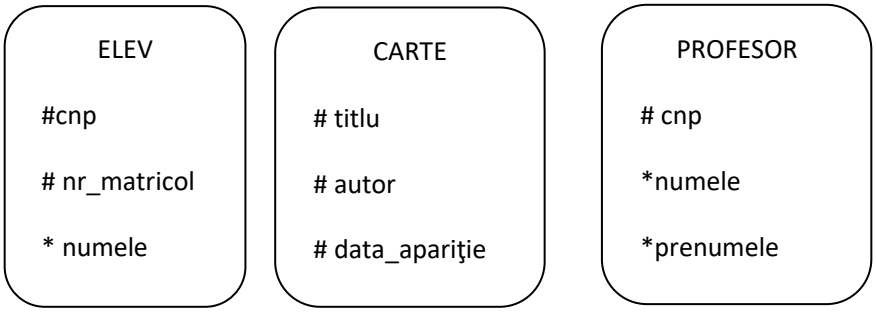


Figura 1.4. Entităţi şi atributele lor

1.5.2. Relaţii între entităţi

Relaţia între două entităţi este o conexiune, o corelaţie, o legătură, o interacţiune între instanţele entităţilor şi care are semnificaţie pentru afacerea modelată. (Exemplu:Un client poate plăti una sau mai multe facturi; Interacţiunea dinte entităţile client şi factură este indicată de verbul „a plăti"). Orice relaţie este bidirecţională: acţiunea este privită din punctul de vedere a ambelor entităţi (Clientul poate să plătească mai multe facturi şi fiecare factură trebuie plătită de un client)

Caracteristicile relaţiei:

1. **Numele** relaţiei este un verb: a avea, a comanda, a înregistra
2. **Opţionalitatea :** - relaţia obligatorie este reprezentată de cuvântul „trebuie"
   - relaţie opţională este reprezentată de cuvântul „poate"

3. **Cardinalitate** indică numărul de instanţe dintr-o entitate care relaţionează cu o singură instanţă din a doua entitate.

Reprezentarea grafică a relaţiilor

Într-un ERD o relaţie se reprezintă printr-o linie care uneşte cele două entităţi. Pentru că o relaţie este bidirecţională, linia este compusă din două segmente distincte, câte unul care pleacă de la fiecare entitate. Tipul segmentului care plecă de la o entitate (linie continuă sau punctată) reprezintă opţionalitatea relaţiei. La cealaltă entitate se reprezintă cardinalitatea printr-o liniuţă simplă sau trei linii unite într-un punct (picior de cioară).

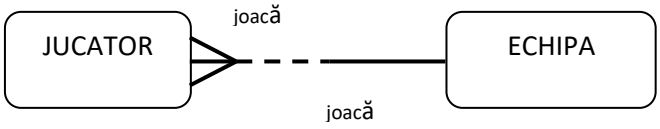
Tabelul 1.1 Reprezentarea grafică a caracteristicilor relaţie între două entităţi

Caracteristica relaţiei	Valoarea caracteristicii	Simbolul grafic
Numele relaţiei	Un verb	Se scrie deasupra relaţiei
Opţionalitatea Se reprezintă imediat după entitate (dreptunghiul cu colţuri rotunjite)	Relaţie obligatorie (TREBUIE)	Linie continuă Linie continuă
	Relaţie opţională (POATE)	Linie întreruptă
Cardinalitatea	Una şi numai una	Linie simplă
	Una sau mai multe	Picior de cioară

Algoritmul de formulare (de citire) a relaţiei intre două entităţi indică modul în care este citită reprezentarea grafică a două entităţi şi a relaţiei dintre ele. O relaţie este bidirecţională astfel încât trebuie să citim de la stânga la dreapta dar şi de la dreapta la stânga:

1. O/Un
2. Numele entităţii A
3. Opţionalitaea (trebuie/poate)
4. Numele relaţiei (verbul)
5. Cardinalitatea (una şi numai una/una sau mai multe)
6. Numele entităţii B

Tabel 1.2. Exemple de formulare a relaţiilor dintre două entităţi

	1.Un **JUCĂTOR** poate **să joace** la o singură **ECHIPĂ** 2. Într-o **ECHIPA** trebuie să joace unul sau mai mulţi **JUCĂTORI**
	1. O **EDITURĂ** trebuie să publice una sau mai multe **CĂRȚI** 2 O **CARTE** poate să fie publicată de o singură **EDITURĂ**

Tipuri principale de relaţii între entităţi

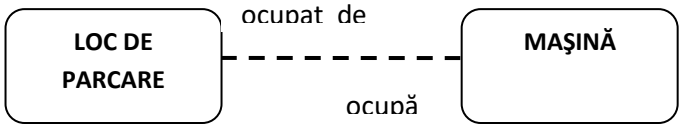
În funcţie de cardinalitatea relaţiei se disting următoarele tipuri

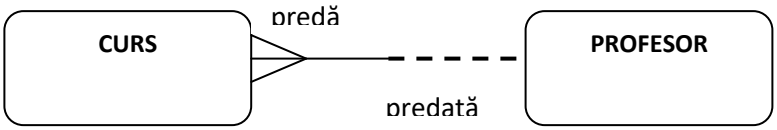
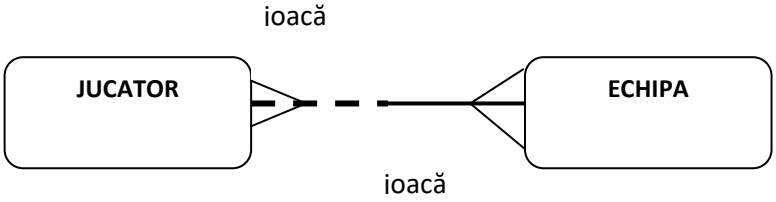
- **Unu la unu (1:1)** o instanţă dintr-o entitate este în relaţie cu o singură instanţă din cealaltă entitate;
- **Unu la mai mulţi (1:M)** o instanţă dintr-o entitate este în relaţie cu mai multe instanţe din cealaltă entitate;
- **Mai mulţi la mai mulţi (M:M)** mai multe instanţe dintr-o entitate sunt în relaţie cu mai multe instanţe din cealaltă entitate;

1.5.3. Modelarea relaţiilor 1:1, 1:M, M:M

Modelarea unei relaţii între două entităţi se realizează în funcţie de tipul relaţiei şi ţinând cont de regulile de reprezentare din tabelul 1.1.

Tabel 1.3. Tipuri de relaţii care se stabilesc între două entităţi

• **One to one 1:1 (unu la unu)** unei instanţie dintr-o entitate îi corespunde o instanţă din entitatea cu care se află în relaţie Se modelează această situaţie prin transformarea uneia din entităţi într-un atribut al celeilalte entităţi	 Un **LOC DE PARCARE** poate să fie ocupat de o **MAŞINĂ** O **MAŞINĂ** poate ocupa un singur **LOC DE PARCARE**

· **Oane to many 1:M (unu la mai mulţi)** unei instanţie dintr-o entitate îi corespunde mai multe instanţe din entitatea cu care se află în relaţie. Sunt cele mai întâlnite tipuri de relaţii.	 Un **CURS** trebuie predat de un **PROFESOR** Un **PROFESOR** poate preda mai multe **CURSURI**
· **Many to many M:M (multe la mai multe)** o instanţe din entitatea A este în relaţie cu mai multe instanţe din entitatea B şi o instanţe din entitatea B este în relaţie cu mai multe instanţe din entitatea A. Acest tip de relaţie trebuie să nu existe în baza de date.	 Un **JUCĂTOR** poate să joace la mai multe **ECHIPE** La o **ECHIPĂ** trebuie să joace mai mulţi **JUCĂTORI**

1.5.4. Algoritmul de rezolvare a relaţiilor many to many

Relaţiile de acest tip nu trebuie să apară în baza de date. Exemplu: un student urmează mai multe cursuri şi pentru fiecare curs primeşte o notă. Un curs este audiat de mai mulţi studenţi care primesc o notă la examen. Dacă entitatea CURS primeşte atributul „notă", nu vom şi cărui student corespunde valoarea atributului, dacă entitatea STUDENT primeşte atributul „notă" nu vom şti cărui curs îi corespunde valoarea atributului

Această situaţie se rezolvă folosind algoritmul „**entitate de intersecţie":**

Pas 1. introducem o entitate nouă numită entitate de intersecţie

Pas 2. legăm entitate de intersecţie de cele două entităţi iniţiale prin două relaţii one to many astfel: la entitatea de intersecţie este partea cu many şi relaţiile sunt obligatorii, iar în partea entităţilor iniţiale este partea one şi se păstrează opţionalitatea relaţiilor iniţiale

Pas 3. adăugarea de atribute entităţii de intersecţie şi identificatorului unic format din identificatorii unici ai entităţilor iniţiale la care puteţi adăuga un identificator unic propriu.

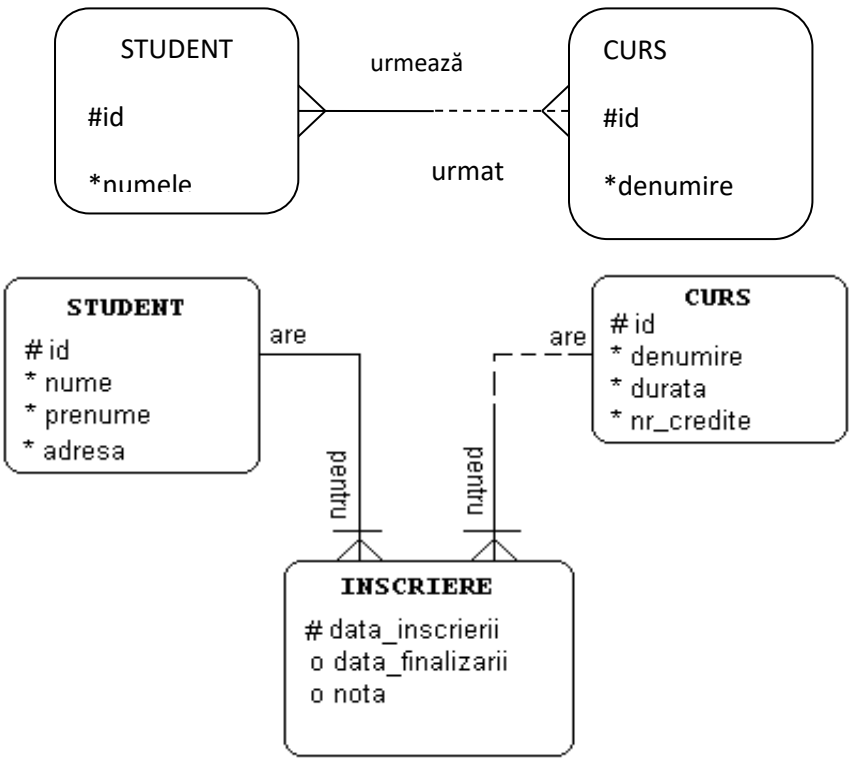


Figura 1.5. Rezolvarea relaţiei many to many

Studiu de caz: O şcoală de arte pune la dispoziţia studenţilor mai multe cursuri. Un curs are o denumire, durată, este predat de un profesor, este programat pentru o anumită zi şi un anumit interval orar, are un preţ. Un student se poate înscrie la unul sau mai multe cursuri. La un curs pot participa mai mulţi studenţi. Pentru a ţine evidenţa studenţilor înscrişi şi a cursurilor, proiectăm două entităţi STUDENT care va reţine datele fiecărui student înscris şi CURS, care ţine evidenţa cursurilor Un student trebuie să primească un calificativ pentru fiecare curs. Unde introducem atributul calificativ?

Este imposibil să introducem atributul „calificativ" la entitatea STUDENT pentru studenţii înscrişi la mai multe cursuri, pentru că o instanţă din entitatea STUDENT este în relaţie cu mai multe instanţe din entitatea CURS şi invers, este imposibil să introducem atributul „calificativ" la entitatea CURS pentru că un curs este frecventat de mai mulţi studenţi şi nu ştim cărui student îi

aparţine calificativul. Rezolvarea relaţiei many to many constă în introducerea entităţii de intersecţie numită ÎNSCRIERE

1.5.5. Modelarea relaţiei barate (relaţie de determinare univocă)

Relaţia barată indică faptul că o entitate preia identificatorul unic de la cealaltă entitatea a relaţiei şi este reprezentată printr-o linie perpendiculară pe relaţia entităţii care preia identificatorul unic (figura 1.5.). Entităţii ÎNSCRIERE are identificatorul unic compus din: identificatorul unic al entităţii STUDENT, id_student şi identificatorul unic al entităţi CURS id_curs şi data_înscriere Această combinaţie identifică în mod unic fişa de înscriere a unui student astfel încât o notă ceorespunde unui singur student.

1.5.6. Modelarea relaţii non transferabile (relaţie de condiţionare biunivocă) Spunem că o relaţie între două instanţe a două entităţi este non transferabilă dacă, odată stabilită, nu mai poate fi modificată. Exemplu

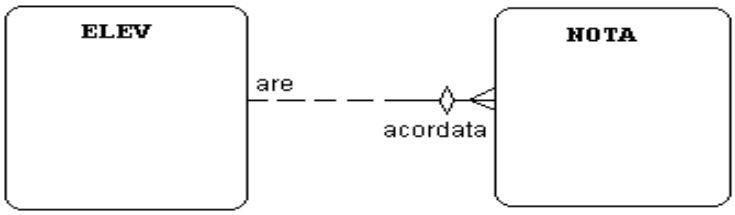


Figura 1.6. Relaţie non transferabilă

Nontransferabilitatea unei relaţii se reduce la faptul că valorile cheii străine corespunzătoare relaţiei respective nu pot fi modificate. Condiţia de nontransferabilitate a unei relaţii este asigurată prin program. De aceea trebuie să documentăm această restricţie. În ERD o relaţie nontransferabilă se notează cu un romb pe linia corespunzătoare relaţiei, entităţii a cărei cheie străină nu este permis să o modificăm (adică în partea cu many a unei relaţii one-to-many).

1.5.7. Modelarea relaţii recursive (modelarea structurilor ierarhice)

Relaţiile recursive modelează structuri ierarhice de tipul: structura de personal a unei instituţii, organigrama unei firme. Fiecare tip de angajat din figura1.7. are ca atribute: nume, prenume, adresă, telefon, email, data naşterii etc. Un angajat conduce unul sau mai mulţi angajaţi şi un angajat este condus de unul sau mai mulţi angajaţi.

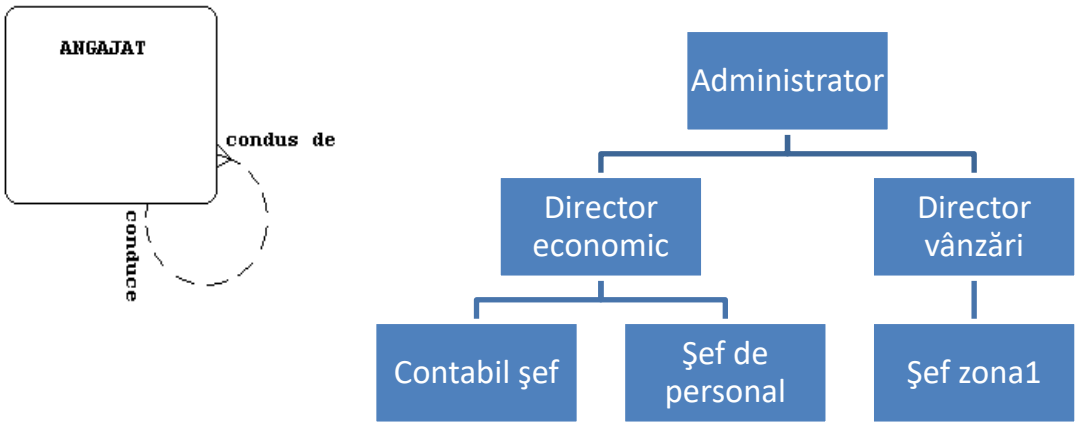


Figura 1.7. Relaţie recursivă

Modelarea acestei structuri se face cu ajutorul unei singure entităţi numită **ANGAJAT**. Aşadar vom avea o relaţie de la entitatea ANGAJAT la ea însăşi.

1.5.8 Modelarea tipurilor şi subtipurilor (modelarea clasificărilor)

În lumea reală obiectele sunt de obicei clasificate. Scopul clasificării pe grupe este de a evidenţia asemănările şi deosebirile. Astfel vorbim despre animale vertebrate şi nevertebrate. Orice entitate poate fi clasificată

Studiu de caz: La modelarea unei afaceri care necesită urmărirea plăţilor de la clienţi observăm că un client poate plăti cu card de credit sau cu cec sau cash. Toate modalităţile de plată au atribute comune: data plăţii şi suma plătită. Dar numai cardul va avea ca atribut nr_card. Pentru a modela aceste clasificări a modalităţilor de plată vom crea o entitate numită MOD DE PLATA şi subentităţile CARD_ DE_ CREDIT, C.E.C CHASH, ALTA (entitate suplimentar introdusă pentru eventuale, noi modalităţi de plată).

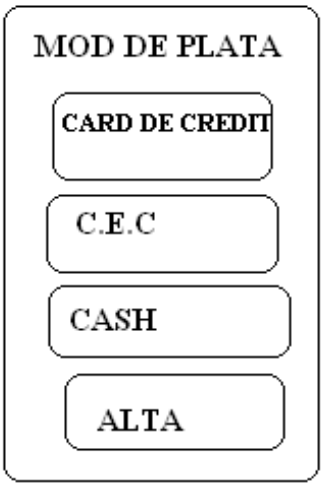


Figura 1.8. Clasificarea modalităţilor de plată

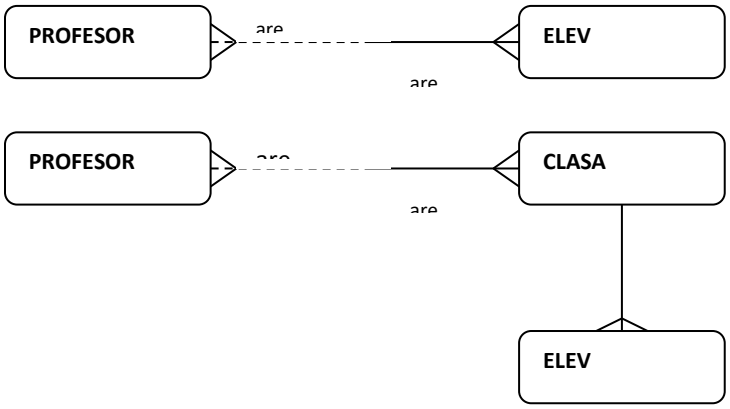
Tipul principal (supertipul) este entitatea care are atribute generale care se regăsesc la nivelul unei clasificări în acest caz entitatea MOD DE PLATA este supertipul.

Subtip sau o sub entitate, o clasificare a unei entităţi care are atribute comune cu entitatea generală, dar are şi atribute proprii, speciale, care nu se regăsesc în celelalte clase.

Subtipurile se reprezintă ca entităţi în interiorul entităţi supertip. Atributele şi relaţiile comune tuturor subtipurilor se vor reprezenta la nivelul supertipului, sau superentităţii şi vor fi moştenite de către subtipuri. Un subtip poate avea la rândul său alte subtipuri incluse.

Reguli ce trebuie respectate la modelarea subtipurilor şi supertipurilor

- Fiecare instanţă a supertipului este de asemenea instanţă unui subtip adică un subtip trebuie să acopere toate cazurile posibile de instanţe ale supertipului.
- Este necesar includerea un subtip cu numele ALTUL (OTHER) pentru a asigura că subtipurile sunt epuizate, că sunt acoperite toate instanţele tipului principal şi a permite viitoare dezvoltări ale modelului.
- Subtipurile trebuie să se excludă reciproc. Exemplu: Angajatul unei echipe de baschet nu poate să fie şi antrenorul şi medicul echipei (figura 1.9.)
- **Relaţii redundante (inutile)**



Introducerea entităţii **CLASA** elimină relaţia inutilă dintre entităţile **PROFESOR** şi **ELEV** deoarece putem determina profesorii care predau unui elev, aflând profesorii clasei care predau la clasa în care este înscris elevul

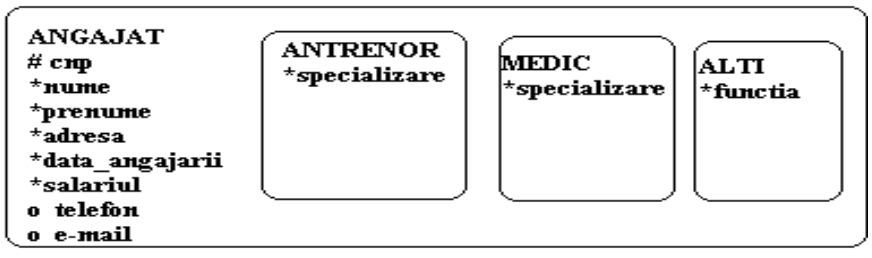


Figura 1.9. Modelarea clasificărilor (tipuri şi subtipuri)

1.5.9. Modelarea arcelor (relaţii exclusive)

Arcele modelează relaţiile exclusive (care se exclud reciproc), Dintr-un grup de relaţii, la un moment dat, doar una dintre ele poate avea loc. Un cont la o bancă trebuie deschis fie de o persoană fizică, fie de o firmă. Relaţiile exclusive sunt reprezentate în ERD printr-un arc peste relaţiile reprezentate pentru cele două entităţi. (Figura 1.9.)

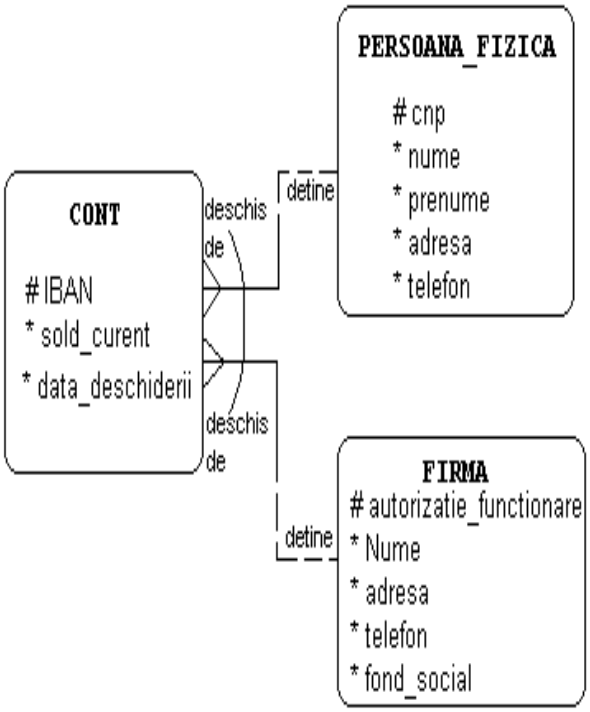


Figura 1.9. Modelarea arcelor

Reguli de reprezentare a arcelor:

- Relaţiile arcului au aceeaşi opţionalitate;
- O entitate poate avea mai multe arce;
- O relaţie aparţine unui singur arc
- Relaţiile arcului pleacă din aceeaşi entitate;

Tipuri de relaţii exclusive (arce):

- relaţii exclusive obligatorii relaţiile ce fac parte din arc sunt obligatorii, ceea ce înseamnă că de fiecare dată, una dintre relaţii are obligatoriu loc. (Figura 1.9.)
- relaţii exclusive opţionale relaţiile ce fac parte din arc sunt opţionale pentru că există varianta ca pentru o instanţă a entităţii căreia aparţine arcul să nu aibă loc nici una din relaţiile exclusive (Figura 1.10.)

Regulile unei şcolii prevăd ca un elev să participe la o singură activitate extraşcolară. Un elev optează să facă parte din echipa de fotbal, sau să participe la cercul literar sau la

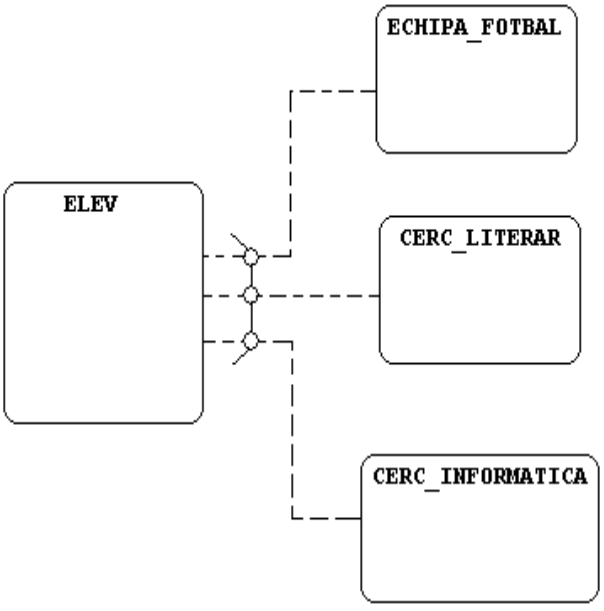
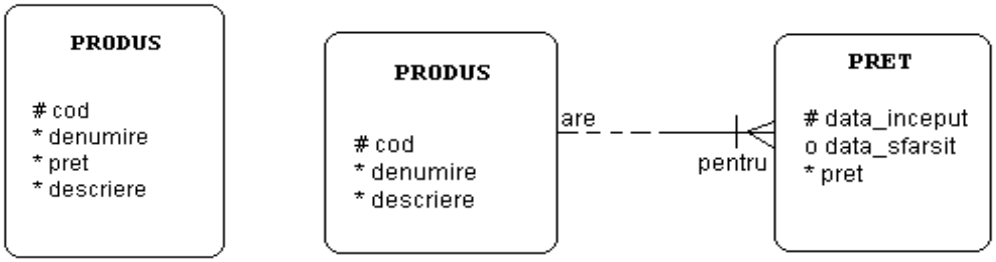


Figura 1.10. Relaţii exclusive opţionale

cercul de informatică sau la nici o activitate

1.5.10. Modelarea datelor istorice

Studiu de caz: Preţul unui produs se poate modifica în timp. Dacă nu ne interesează decât preţul actual al fiecărui produs modelarea se face ca în Figura1.11. a), iar dacă este important să reţinem un istoric al preţurilor pentru fiecare produs, atunci atributul preţ se va transforma într-o entitate (Figura1.11. b)



a) Figura 1.11. Modelarea datelor istorice b)

Atributul data_sfârşit este opţional, deoarece data până la care este valabil preţul curent al unui produs nu este de obicei cunoscută şi poate să nu apară în tabelă o valoare.

Studiu de caz: Proiectarea bazei de date pentru o BIBLIOTECĂ.

Explicaţie:.Într-o primă fază vom obţine o relaţie many-to-many între entităţile CARTE şi CITITOR..(Figura 1.11a). Fiecare carte poate fi împrumutată de mai mulţi cititori (evident nu în acelaşi timp), şi fiecare cititor poate împrumuta mai multe cărţi. Rezolvăm relaţie many-to-many Identificatorul unic este acum combinaţia atributelor cod_carte şi data_împrumut. Un cititor poate împrumuta două cărţi la aceeaşi dată, combinaţia celor două coloane,

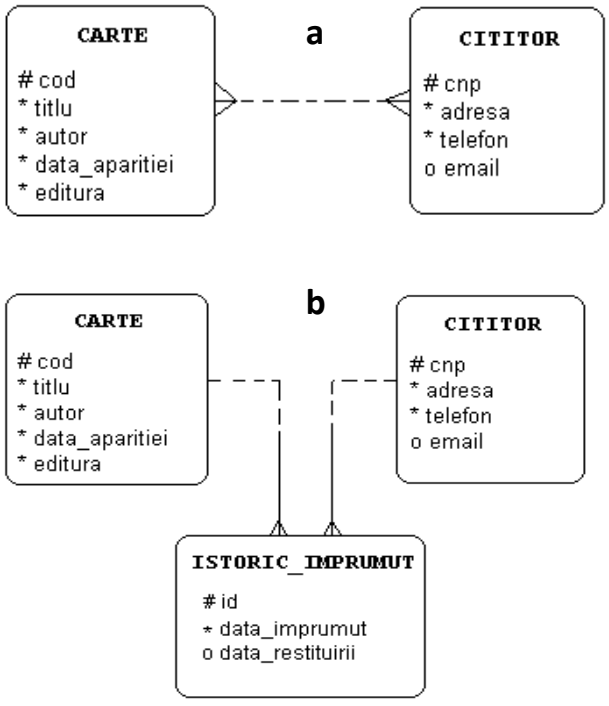


Figura 1.11. Modelarea datelor istorice

(cod_carte şi data_împrumut) pentru fiecare din cele două înregistrări este unică.

Bararea automată a celor două relaţii dinspre entitatea de intersecţie nu este întotdeauna o soluţie corectă. Pentru a evita aceste complicaţii putem recurge la introducerea unei identificator artificial în entitatea de intersecţie.

1.6. Normalizarea datelor

1.6.1 Identificatorul unic (UID) este un atribut sau o combinaţie de atribute şi/sau o relaţie ale căror valori dau unicitate unei instanţe dintr-o entitate şi permit utilizatorului să regăsească în mod unic o instanţă într-o entitate pentru că are valori unice şi nenule.

Identificatori unici formaţi din atribute:

- UID simplu este format numai dintr-un atribut
- UID artificial un atribut artificial creat cu scopul identificării unice într-un sistem
- UID compus este un identificator unic format din două atribute

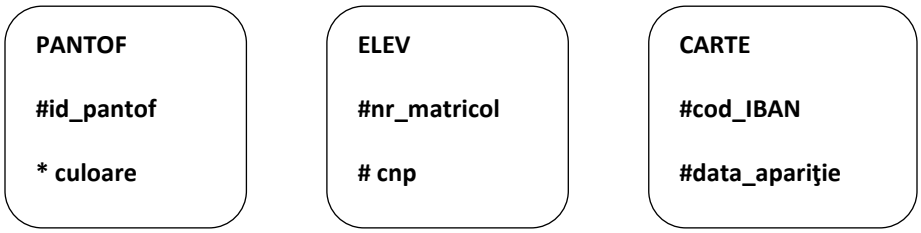


Figura 1.12 Tipuri de identificatori unici

Identificatori unici formaţi din atribute şi relaţii:

- Identificator unic format din două relaţii barate

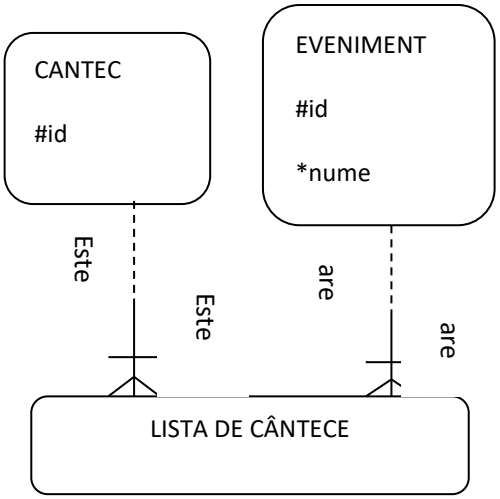


Figura 1.13. UID format din două relaţii barate

UIDul entităţii de intersecţie LISTA DE CÂNTECE este compus din UID urile entităţii iniţiale CANTEC şi EVENIMENT (din relaţii). Barele din relaţie arată ca identificatorul unic entităţii de intersecţie va fi format din identificatorii unici ai entităţilor iniţial aflate în relaţia M:M.

Este o indicaţie importantă transformarea entităţii într-un tabel(maparea diagramei)

• Identificator unic format din relaţie şi un atribut

Studiu de caz: Un cont este deschis la o bancă, o bancă poate deschide mai multe conturi. Poate să fie emis un cont cu acelaşi număr şi de o altă bancă. Pentru a asocia în mod unic un număr de cont cu banca care l-a emis trebuie să barăm relaţia spre entitatea cont ceea ce va determina, pe cale de consecinţă, ca identificatorul unic al entităţii CONT să fie format din combinaţia: nr_cont şi relaţia barată (id_bank)(Figura 1.13.)

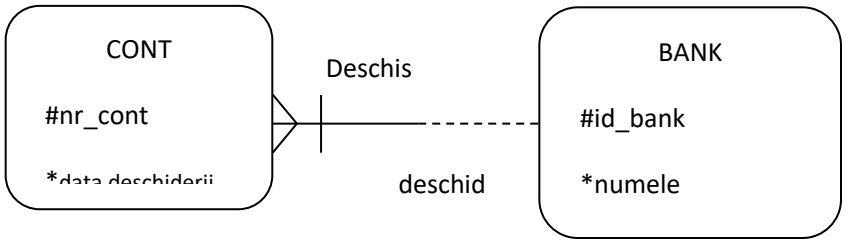


Figura 1.14 UID compus din atribut şi o relaţie barată

Transferurile interbancare au întotdeauna nevoie de un număr de transfer adiţional care este de fapt numărul de cont al băncii UID CONT=id_bank (relaţia)+nr_cont

• Identificator unic –compus din relaţii şi identificator unic artificial

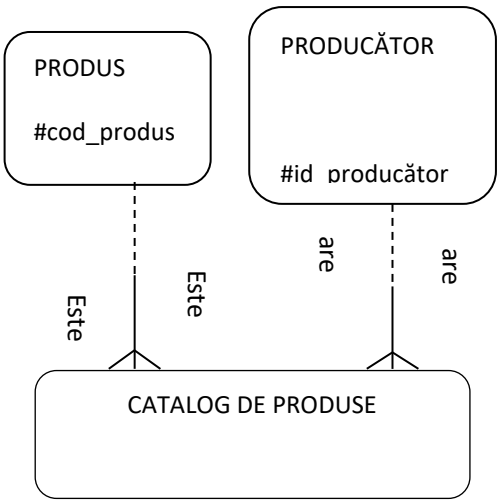


Figura 1.15 UID format din relaţie şi identificator artificial

d) **UID-candidat** Uneori este necesar mai mult de un UID Când comandaţi un produs de pe un site, ţi sa va da un cod unic şi ţi se va cere adresa de e_mail Fiecare dintre aceste două atribute te identifică în mod unic şi sunt UID candidate. Dar dintre cei doi, numai unul va devenii UID principal, iar celălalt este UID secundar

Este posibil ca o entitate de intersecţie să folosească un atribut „aritficial" ca UID Fiecare **PRODUCĂTOR** poate produce unul sau mai multe **PRODUSE**, iar fiecare **PRODUSE** poate fi realizat de unul sau mai mulţi **PRODUCĂTOR.** Ne aflăm în situaţia unei relaţii M:M, a cărui rezolvare este entitatea de intersecţie *CATALOGDE PRODUSE*. Un produs din catalog este identificat unic după id_producător, cod_produs, nr_catalog

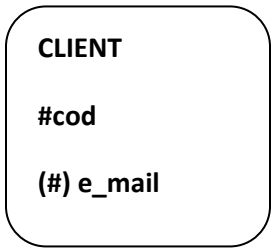


Figura 1.16. UID principal şi secundar

1.6.2. Forme de normalizare -Tehnica normalizării relaţiilor

Proiectarea eficientă a bazelor de date relaţionale presupune: definirea atributelor, gruparea lor în tabele, stabilirea legăturilor dintre ele şi respectarea regulilor de integritate a datelor care garantează că datele introduse în baza de date sunt corecte şi valide. Se elimină astfel generarea anomaliilor la actualizarea, modificarea sau ştergerea unor date.

Normalizarea este o tehnică de proiectare a bazelor de date care are ca obiectiv respectarea **regulilor de integritate** la proiectarea bazelor de date:

- **integritatea entităţilor** (la nivel de tabelă)- o coloană care face parte din cheia primară trebuie să aibă o valoare unică şi nu poate avea valoarea NULL

• **integritatea referenţială** (la nivel de relaţie)- fiecare valoare a cheii străine să corespundă unei valori a cheii primare din tabela de referinţă

• **integritatea de domeniu** valorile introduse într-o coloană să aparţină unui anumit domeniu

Edgar Frank Codd părintele modelului relaţional, a definit iniţial trei reguli de normalizare cunoscute sub denumirea de forme normale şi notate prin 1NF, 2NF şi 3NF.

Tehnica de normalizare a relaţiilor constă în a verifica dacă entităţile respectă cele trei forme normale prin eliminarea unui anumit tip de dependente nedorite :

• dependente multivaloare (FN1),

• dependente parţiale (FN2)

• dependente tranzitive (FN3)

(1FN) Prima formă normală elimină atributele non atomice şi repetitive

Enunţ: O entitate respectă prima formă normală dacă şi numai dacă atributele sunt atomice, adică conţin o singură informaţie.

• nu există atribute cu valori multiple (atribute compuse);

• nu există atribute sau grupuri de atribute care se repetă. (atribute repetitive)

Algoritmul de normalizare 1FN este:

• **identificăm** atributul care nu respectă 1NF (are valori multiple/atribute repetitive)

• **construim** o entitate adiţională cu numele atributului care se repeta

• **relaţionăm** entitatea iniţială cu entitatea adiţională printr-o relaţie 1:M, relaţia este many spre entitatea nou creată, adiţională

Studii de caz 1. **Atribute compuse**. (non_atomice)

• Atributul cu valori multiple ***data_calendaristică,*** alcătuit, din câmpurile: *Ziuă*, *Lună*, *An*. SGBD-ul Oracle prezintă tipul de dată DATE prin care se gestionează simultan toate cele trei elemente toate cele trei elemente; prin funcţii de conversie se pot obţine: ziua (DAY), luna (MONTH), anul (YEAR), ba chiar calcula numărul de zile, luni, ani dintre două date calendaristice şi alte minunăţii. Ca să nu mai vorbim de tipul DATETIME

• Un alt exemplu celebru privind atribute compuse (non-atomice) este Adresa**.** Fiind alcătuită din Stradă, Număr, Bloc, Scară, Etaj, Apartament, discuţia despre atomicitatea adresei pare de prisos, iar descompunerea sa imperativă. Din nou însă trebuie să ne raportăm la

obiectivele bazei. Pentru o baza de date adresa interesează numai la nivel general pentru Romtelecom, preluarea separată a fiecărui element constituent al adresei este vitală.

• Alte exemple de atribute ce pot fi considerate, în funcţie de circumstanţe, simple sau compuse: DataOperaţiuniiBancare (Data + Ora), BuletinIdentitate (Seria+Număr), NrÎnmatriculareAuto (privit global, sau pe cele trei componente: număr, judeţ, combinaţie trei de litere).

Studii de caz 1 **Atribute repetitive şi urmările lor**

O entitate, relaţie (tabelă) în 1FN nu trebuie să conţină atribute care se repetă ca grupuri. Un grup repetitiv este un atribut sau grup de atribute dintr-o entitate care apare cu valori multiple pentru o singură apariţie a cheii primare sau altfel formulat:.toate liniile unei tabele trebuie să conţină acelaşi număr de atribute.(coloane)

Exemplu de normalizare:

Să considerăm entitatea ELEV referitoare la notele elevilor unei clase la disciplinele studiate. Un elev studiază mai multe discipline şi la o disciplină poate avea mai multe note. Entitatea elev nu respectă 1FN există atribute cu valori multiple . Vom crea o nouă entitate în care vom introduce disciplina şi nota la disciplina respectivă

În acest fel un elev la o disciplină poate avea oricâte note, singura restricţie conform acestui model fiind că un elev nu va putea primi în aceeaşi zi la aceeaşi materie mai multe note.

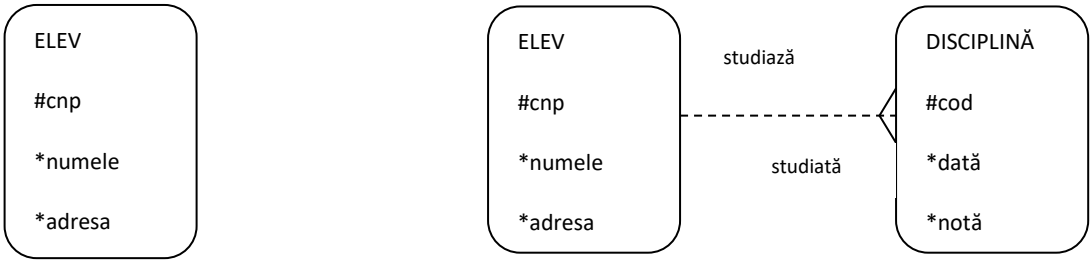


Figura 1.16 Normalizare 1FN

(2FN) A doua formă normală elimină dependenţele parţiale

Enunţ: O entitate se găseşte în a doua formă normală dacă şi numai dacă

• se găseşte în prima formă normală

• orice atribut care nu face parte din UID (unique identifier) va depinde de întregul UID nu doar de o parte a acestuia.

Se aplică entităţilor care au UID-ul compus

Orice entitate aflată în prima formă normală care are UID simplu (format dintr-un singur atribut) este automat în a doua formă normală

Algoritmul de normalizare 2FN (transformă UID compus în UID simplu)

• **construim** o entitate adiţională care va prelua toate atributele care nu depind de întreg UID-ul.

• **relaţionăm** entitatea iniţială cu entitatea adiţională printr-o relaţie 1:M (one to many), relaţia este many spre entitatea nou creată, adiţională

Exemplu

Entitatea DEPARTAMENT care memorează toţi angajaţii unu departament şi are un UID compus din id_dep şi id_angajat nu respectă 2FN pentru că atributele data_naşterii şi adresa depind doar de id_angajat nu şi de id_dep (identificatorul departamentului)

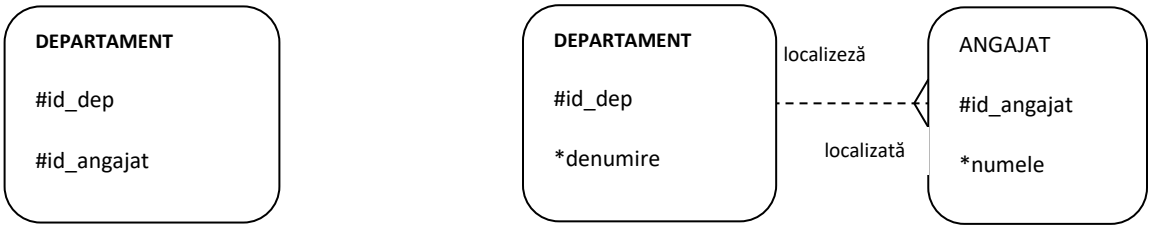


Figura 1.17 Normalizare 2FN

(3FN) A treia formă normală 3FN-elimină dependenţele tranzitive

Enunţ: O entitate se găseşte în a treia formă normală dacă şi numai dacă

• se găseşte în a doua formă normală

• nu are dependenţe tranzitive (un atribut care nu este parte a UID depinde numai de atribute non-UID, care nu sunt parte a UID)

Algoritmul de normalizare (elimină dependenţele tranzitive)

• **construim** o entitate cu atributele dependente tranzitiv

• **relaţionăm** entitatea iniţială cu entitatea adiţională printr-o relaţie 1:M (one to many), relaţia este many spre entitatea iniţiale

Studiu de caz

În entitate **LOCALITATE** atributele Stema şi numele prefectului sunt dependente tranzitiv, pentru că depind numai de atributul judeţ care nu este parte a UID.

Aplicăm tehnica de normalizare şi mutăm atributele dependente tranzitiv într-o entitate cu numele JUDEŢ care va fi în relaţia 1:Mcu entitatea iniţială

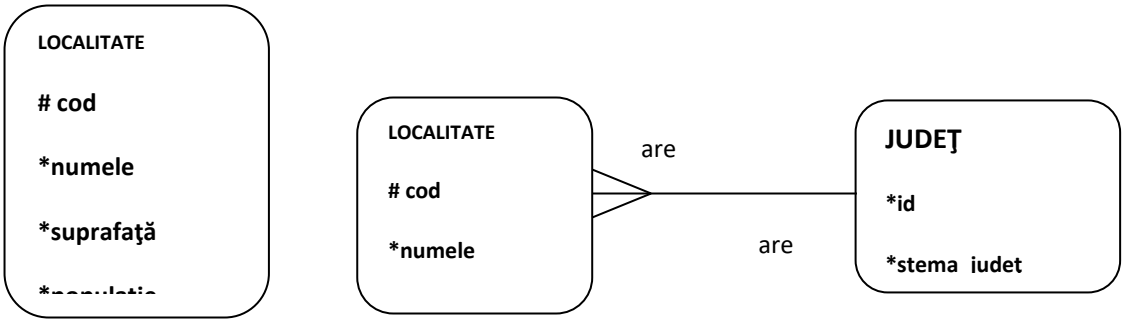


Figura 1.18 Normalizare 2FN

Proiectarea bazelor

1. Proiectarea scheme conceputuale iniţiale prim metoda reprezentărilor grafice ERD-ului

2 .Definirea schemei relaţiilor (mulţimea de relaţii din baza de date) si a restricţiilor de integritate prin tehnica normalizării relaţiilor

3. Maparea adică descrierea schemei conceptuale în limbajul de descriere a datelor utilizat de SGBD-ul relaţional ce se utilizează).

1.7. Proiectarea modelului fizic al bazei de date – Maparea

Modelul fizic al bazei de date se obţine din modelul conceptual prin mapare

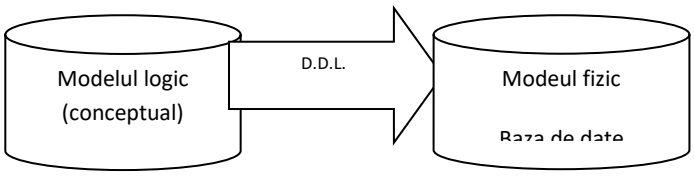


Figura 1.18 Maparea

Tehnica de mapare : înseamnă schimbarea terminologiei, un element grafic din modelul conceptual devine un obiect al modelului fizic. Definirea obiectelor se realizează cu ajutorul limbaj de descriere a datelor (DDL), utilizat de SGBD-ul Oracle

1.7.1 Crearea tabelelor prin maparea entităţilor

Elemntele grafice din modelul conceptual se tranformă în obiectele modelului fizic

Tabel 1.4. Transformarea simbolurilor grafice în obiecte

Element grafic în modelul conceptual	Obiect în modelul fizic
O entitate	Un tabel
O instanţă	Un rând
Un atribut	O coloană
Identificatorul unic primar UID	O coloană numită cheie primară
O relaţie dintre două entităţi	O coloană numită cheie secundară
Identificatorul unic secundar	O coloană numită cheie unică

Cheia primară este o coloană sau o combinaţie de coloane care identifică în mod unic rândurile dintr-o tabelă

Cheie secundară este cheia primară din tabela de referinţă, o coloană suplimentară introdusă în tabela corespunzătoare entităţii din partea many a relaţiei şi corespunde coloanei cheia primară a tabelei corespunzătoare entităţii din partea one a relaţiei

Diagrama de tabelă conţine informaţii despre structura unei tabele:

Tabel 1.5. Structura diagramei de tabelă

Numele tabelei (numele entităţii la plural)			
Numele coloanei este numele atributului	Tipul de dată	Opţionalitate #,*, o	Tipul de cheie Pk, Fk

Tabel 1.6. Tipurile de date Oracle

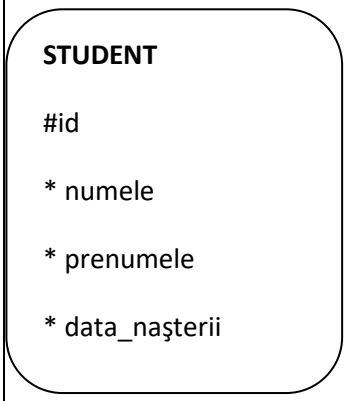
Tipul de date	Descriere	Dimensiune Maximă
VARCHAR2	Şir de caractere de lungime variabilă	4000 bytes
CHAR	Şir de caractere de lungime fixă	2000 bytes
NUMBER(p,s)	Număr având p cifre din care s la partea zecimală.	p (precizia) între 1 şi 38 s (scala) între -84 şi 127.
DATE	Dată calendaristică (zz/ll/aa)	De la 1 Ianuarie 4712 BC pana la 31 Decembrie, 9999 AD.
TIMESTAMP	Se memorează data calendaristică, ora, minutul, secunda şi fracţiunea de secundă	Fracţiunea de secundă este memorată cu o precizie de la 0 la 9.
INTERVAL YEAR TO MONTH	perioadă de timp în ani şi luni.	
INTERVAL DAY TO SECOND	memorează un interval de timp în zile, ore, minute şi secunde	
CLOB	Character Large Object	4 Gigabytes
BLOB	Binary Large Object	4 Gigabytes
BFILE	Se memorează adresa unui fişier binar	4 Gigabytes

	de pe disc	

Execiţiu completaţi diagrama de tabelă pentru entitatea STUDENT

Tabel 1.6 Exemplu de mapare a unei entităţi

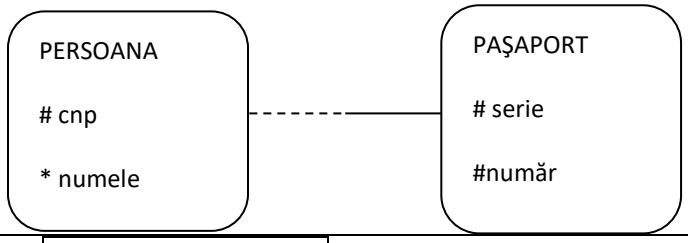
Mapaţi următoarea entitate:



STUDENŢI			
Numele coloanei	Tipul de dată	Opţionalitate	Tipul de cheie
id	Number	Pk	*
Numele	Varchar2	-	*
Prenumele	Varchar2		*
Data_naşterii	Date		*
Telefon	Number		*
adresă	Varchar2		o
e_mail			o

1.7.2. Maparea relaţiilor 1:1 (one-to-one)

Cheia primară a tabelei **PAŞAPOARTE** este cheie străină în cadrul tabelei **PERSOANE**, dar putem proceda la fel de bine şi invers Tabel 1.7 Exemplu Mapare 1:1



.PERSOANE			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
Cnp	Number	*	Pk
Serie_pasaport	Char	*	Fk
Numar_paşaport	Number	*	Fk
Numele	Varchar2	*	-
Prenumele	Varchar2	*	-
Adresa	Varchar2	*	-
Telefon	Number	o	-
e-mail	Char	o	-

PAŞAPOARTE

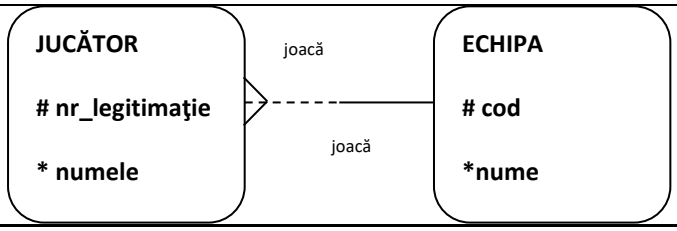
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
nr	Number	*	Pk
serie	char	*	Pk
Data_emiterii	date	*	-
Data_expirării	date	*	-
emitent	Varchar2	*	-

2. Maparea relaţiilor 1:M (one to many)

Regulă: O relaţie 1:M creează una sau mai multe coloane cheie secundară într-un tabel. Coloana cheie secundară este obligatorie sau opţională în funcţie de opţionalitatea realaţiei din partea many

Explicaţie: La maparea introducem în tabela corespunzătoare entităţii de pe partea many a relaţiei, un câmp/câmpuri care se vor numi cheie străină (foreign key) şi reprezintă cheia primară a entităţii de pe partea one a relaţiei;.); dacă relaţia pe partea many este opţională atunci şi coloanele cheii străine vor fi opţionale. Înseamnă că în câmpul cheie străină pot exista valori NULL, câmpul poate rămâne necompletat cu valori. Dacă relaţia pe partea many este obligatorie atunci coloanele ce fac parte din cheia străină vor fi obligatori, coloana nu poate avea valori NULL (trebuie introdusă o valoare)

Studiu de caz1. Tabel 1.8. Maparea relaţiei 1:M



JUCĂTORI

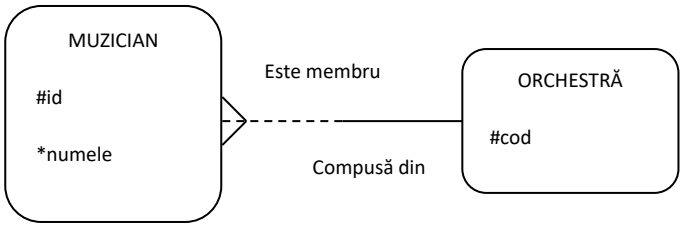
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
nr_legitimaţie	Number	*	Pk
cod_echipă	Number	o	Fk
Numele	Varchar2	*	-
Prenumele	Varchar2	*	-
Adresa	Varchar2	*	-
Telefon	Number	o	-
e-mail	Char	o	-

ECHIPE

Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
cod	Number	*	Pk
numele	Varchar2	*	-

localitate	Varchar2	*	
Emblema	Blob	*	-
Adresa_club	Varchar2	*	-

Studiu de caz2. Tabel 1.9. Maparea relaţiei 1:M



MUZICIENI			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
id	Number	*	Pk
cod_ochestră	Number	o	Fk
Numele	Varchar2	*	-
Prenumele	Varchar2	*	-
Adresa	Varchar2	*	-
Telefon	Number	o	-
e-mail	Char	o	-

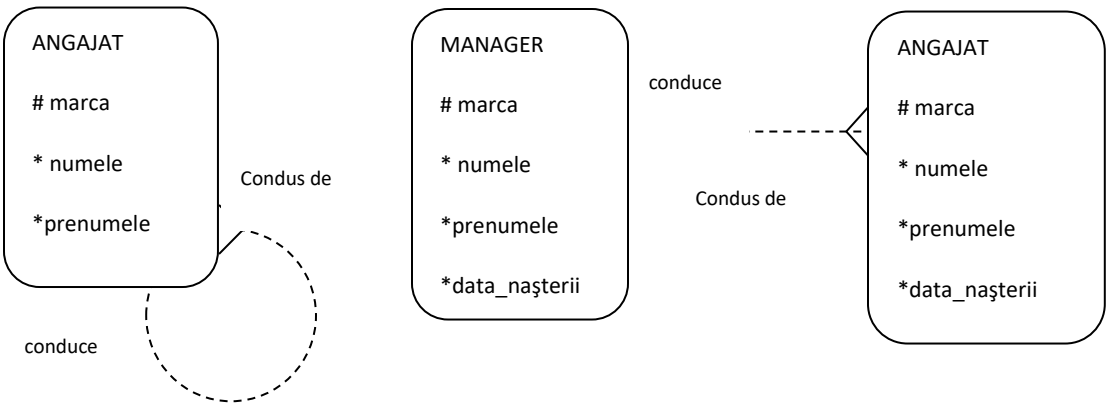
ORCHESTRE			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
cod	Number	*	Pk
numele	Varchar2	*	-

1.7.4. Maparea relaţiilor recursive

a) Relaţia recursivă este fi privită ca o relaţie între două entităţi identice

Aşadar vom introduce în cadrul tabelei ANGAJAŢI, marca managerului său

Figura 1.19 Modelarea relaţiei recursive ca relaţie one to many



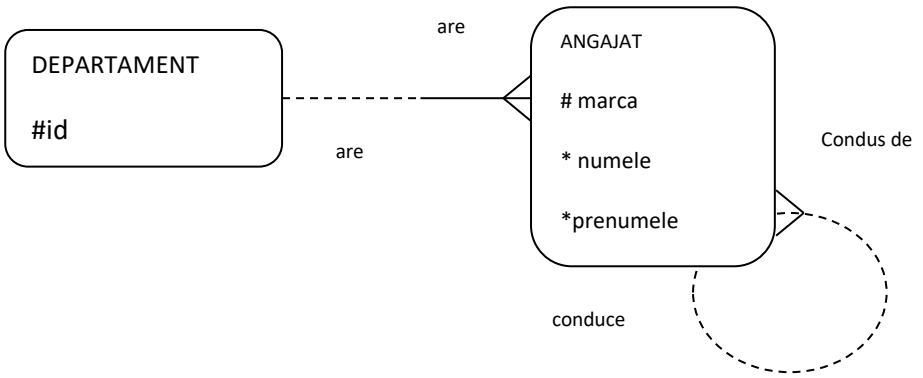
La maparea tabelei ANGAJAT apare un câmp suplimentar cod_manager

Tabel 1.10 Diagrama tabelei ANGAJATI

ANGAJAŢI			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
marca	Number	*	Pk
cod_manager	Number	o	Fk
Numele	Varchar2	*	-
Prenumele	Varchar2	*	-
Adresa	Varchar2	*	-
Telefon	Number	o	-
e-mail	Char	o	-

b) o completarea variantei a) cu entitatea **DEPRTAMENT**

Tabel 1.11. Diagrama tabelei ANGAJATI şi DEPARTAMRNT



ANGAJAŢI			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
marca	Number	*	Pk
cod_manager	Number	o	Fk
id_department	Number	*	Fk
Numele	Varchar2	*	-
Prenumele	Varchar2	*	-
Adresa	Varchar2	*	-
Telefon	Number	o	-
e-mail	Char	o	-

DEPARTAMENTE			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
id	Number	*	Pk
numele	Varchar2	*	-

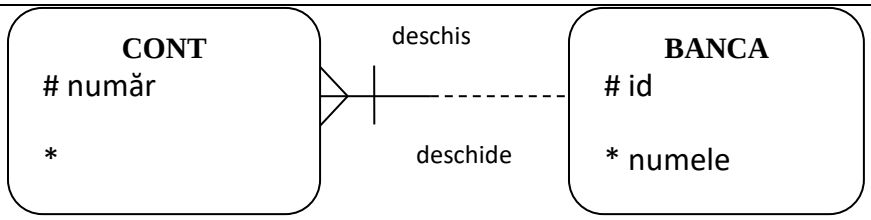
1.7.5. Maparea relaţiilor barate

Bararea unei relaţiile 1:M (bara apare în partea many) indică faptul că identificatorul unic al entităţii din partea many preia şi identificatorul unic al entităţii din partea one. Şi devine UIC compus. La mapare, relaţia şi bararea relaţiei dintre cele două tabele este reprezentată prin

coloana suplimentară, cu o dublă semnificaţie: cheie străină, şi cheie primară (bara indică faptul ca este o parte din cheia primară)

Studiu de caz: în acest exemplu id_bancă este o coloană cheie străină în tabelul **CONTURI** care se referă la o coloană cheie primară a tabelului **BĂNCI** Şi este de asemenea parte a cheii primare din tabela **CONTURI**

Tabel 1.12 Maparea relaţiei barare Diagramele de tabele şi ERD-ul.



CONTURI			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
număr	Number	*	Pk
id_bancă	Number	*	Pk ,Fk
data_deschiderii	Number	*	-
balanţă	bfile	*	-

BĂNCI			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
id	Number	*	Pk
numele	Varchar2	*	-
localitate	Varchar2	*	-

Maparea relaţii barate în cascadă

Ierarhiile pot duce la relaţii barate în cascadă, în care UID-ul entităţii din fruntea ierarhiei este păstrat până la UID-ul entităţii din baza ierarhiei. Pentru entitatea CAMERĂ UID-ul se compune din: nr_cameră, nr_apartament, nr_etaj, id_bloc

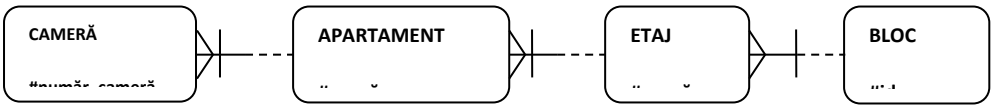


Figura 1.20 Relaţii barate în cascadă

Tabel 1. 13. Maparea relaţiei barate în cascadă Diagramele tabelelor

BLOCURI			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
id	Number	*	Pk
adresă	Number	*	/

ETAJE			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
număr	Number	*	Pk
id_bloc	Number	*	Pk ,Fk

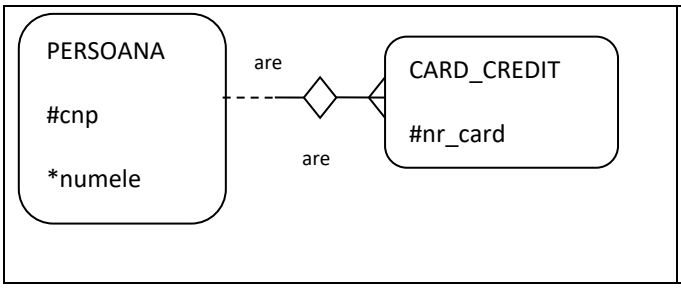
APARTAMENTE			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
număr	Number	*	Pk
nr_etaj	Number	*	Pk ,Fk
nr_bloc	Number	*	-

CAMERE			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
număr	Number	*	Pk
nr_apartament	Number	*	Pk ,Fk
nr_etaj	Number	*	Pk ,Fk
d_bloc	number	*	Pk ,Fk

1.7.6. Maparea relaţiilor nontransferabile

Condiţia de non transferabilitate a unei relaţii implică: valoarile cheii străine iniţial introduse rămând neschimbate. O relaţie nontransferabilă din modelul conceptual din Tabel 1.14., la mapare devine o coloana cheie străină în tabelul entităţii din partea many a relaţiei care nu poate fi modificată (nu poate fi modificată asocierea cnp_pers/nr_card, este unică) În acest caz cheia secundară joacă rol şi de cheie primară unică

Tabel 1.14Diagrama entităţi relaţie şi diagrama de tabelă a entităţii CARED_CREDIT



CARD _CREDIT		
Tipul de cheie	Opţionalitate	Nume coloană
Pk	*	Nr_card
	*	Data_deschidere
Kk	*	Cnp_pers

Pentru realizarea acestei constrângeri este necesară programarea adiţională. Este important să documentăm această regulă pentru ca echipa de programatori să scrie codul potrivit

1.7.7. Maparea arcelor

Pentru a mapa un arc vom crea în tabela căruia îi aparţine arcul, atâtea coloane chei străine câte relaţii există în arcul respectiv

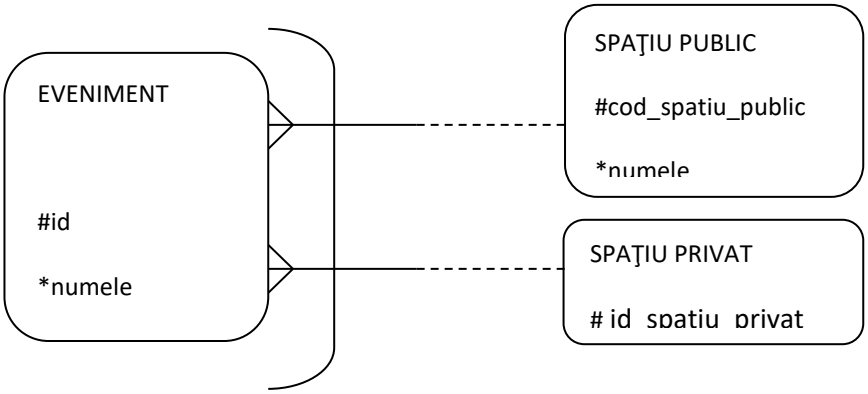


Figura1. 21 Diagrama ER a unui arc

Tabel 1. 14. Diagramele de tabele în formă simplificată

EVENIMENTE		
id		k
numele		
data		
cost		
deacriere		
cod_spatiu_public		k
id_spaţiu_privat		k

SPAŢIU PRIVAT

id_spaţiu_privat	*	Pk
numele	*	-
adresă	*	-

SPAŢII PUBLICE		
cod_spatiu_public	*	Pk
numele	*	-
adresă	*	-
cost_chirie	*	-

1.7.8. Maparea tipurilor şi subtipurilor

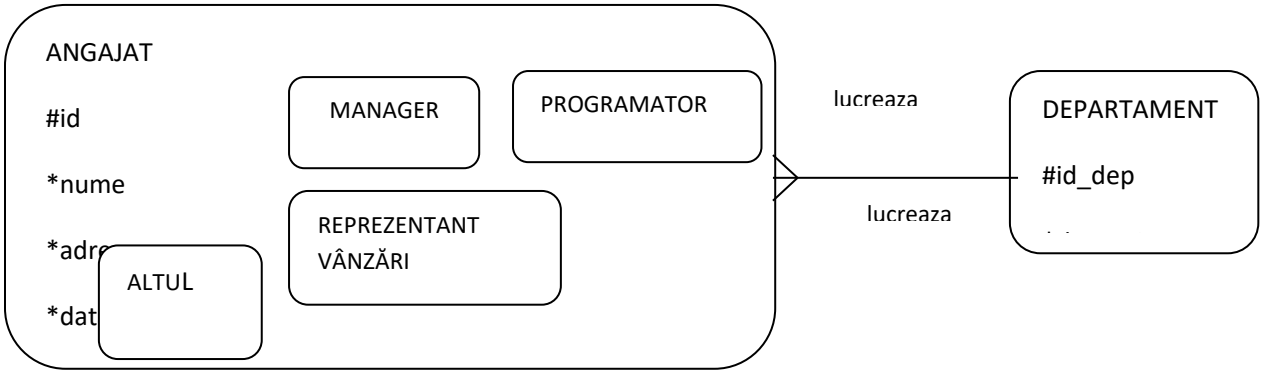


Figura 1.22 Diagrama ER pentru un subtip ANGAJAT

a) Metoda „Tabel unic” se implementarea supertipul ca tabel unic cel mai probabil atunci când: Marea majoritate a atributelor şi relaţiilor sunt la nivelul supertipului şi regulile de afaceri se aplică global

Tehnică Se creează un singur tabel care conţine toate coloanele corespunzătoare atributelor la nivel de supertip dar şi coloanele atributelor subtipurilor

- Coloanele corespunzătoare atributelor supertipului îşi păstrează opţionalitatea
- Coloanele corespunzătoare atributelor obligatorii ale subtipurilor vor devenii opţionale. Pentru a verifica dacă ele sunt completate se stabileşte o regulă de integritate la nivel de înregistrare
- Identificatorul unic al supertipului devine cheie primară

• Identificatorii unici la nivelul subtipului se transformă în coloane chei străine opţionale

• Se introduce un câmp suplimentar tip_angajat, cu ajutorul căruia se identifică subtipul din care face parte angajatul: manager, programator, agent de vânzări

Tabel 1.15. Diagrama de tabelă ANGAJATI

ANGAJAŢI			
Numele coloanei (numele atributului)	Tipul de dată	Opţionalitate	Tipul de cheie
id	Number	*	Pk
nume	Number	*	-
adresă	Varchar2	*	-
data_naşterii	Date	*	-
Salariul	Number	*	-
Id_dep	Number	*	Fk
Bonus	Number	o	
zona	Varchar2	o	Fk
specializarea	Varchar2	*	-
Tip_ang	Numeric	*	-

b) Metoda „tabele pentru fiecare subtip*”*

Se utilizează în cazurile când tabelele sunt supuse unor operaţii diferite: unul pentru modificare, altul pentru interogare, regulile afacerii sunt diferite pentru fiecare subtip, subtipurile au foarte puţine atribute comune

Tehnică Pentru fiecare subtip se realizează un tabel

• în care se introduce câte o coloană pentru atributele supertipului cu opţionalitatea originală şi o coloană cheie străină corespunzătoare coloanei cheie primară

• UID supertipului crează o coloană cheie primară pentru fiecare tabel

• Pentru relaţiile la nivelul supertupului se introduce în fiecare tabel o coloană cheie străină cu opţionalitatea relaţiei

Observaţie: se pot modela subtipurile ca entităţi de sine stătătoare în relaţia arc one to one cu **supertipul.**

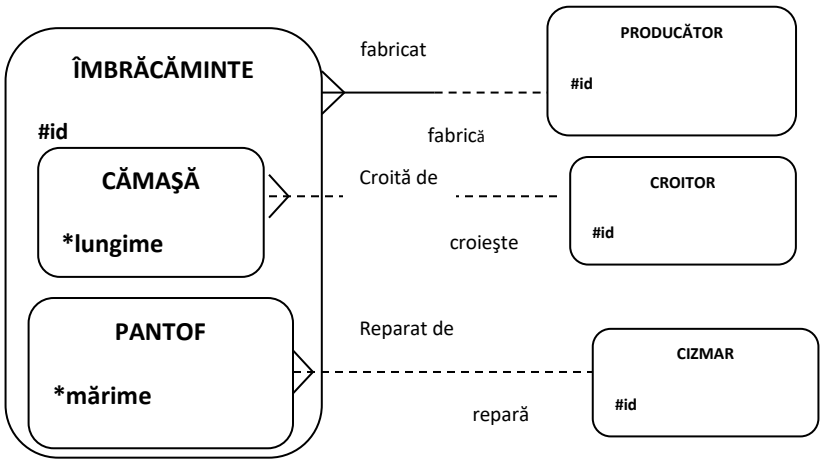


Figura 1.23. Diagrama ER a supertipului îmbrăcăminte

CAPITOLUL II

PROGRAMAREA BAZELOR DE DATE

2.1 Introducere în limbajul SQL

SQL (Structured Query Language -limbajul de interogare structurat) este un limbaj special conceput pentru comunicarea cu bazele de date relaţionale.

Avantajele limbajului SQL

- SQL este un limbaj formal, ne procedural, foarte simplu, dar performant, alcătuit dintr-un număr foarte redus de cuvinte pentru că este proiectat pentru a furniza o modalitate simplă şi eficientă de a comunica cu baza de date, dea efectua operaţiile în baza de date Oracle folosind instrucţiuni;
- S.G.B.D-urile (sistemele de gestiune a bazelor de date - Database Management System - DBMS) recunosc limbajul SQL (DB2, Interbase, MySQL, Oracle, Postgresql, SQL Server, Sybase); învăţând acest limbaj putem reacţiona cu aproape toate bazele de date indiferent de sistemul care le administrează, prin intermediul PHP.
- SQL standard este controlat de comisia de standardizare ANSI (American Naţional Standards Institute - ANSI SQL). Producătorii de S.G.B.D şi-au extins suportul pentru SQL prin adăugarea de instrucţiuni la limbajul standard în scopul realizării unor funcţionalităţi suplimentare şi modalităţi simplificate de efectuare a unor operaţii. Implementările individuale îşi au propriul lor nume (ORACLE are PL-SQL).
- SQL are o structură proprie şi sintaxă, pentru un set de instrucţiuni prin care un programator/administrator de baze de date poate realiza următoarele:

1. interogarea bazei de date asupra informaţiilor;
2. actualizarea, inserarea şi eliminarea datelor
3. crearea, modificarea şi eliminarea obiectelor bazelor de date
4. accesului la baza de date (securitatea sistemului)
5. asigurarea integrităţii şi consistenţei datelor

2.1.1. Elemente de bază ale limbajului SQL

Limbajul SQL comunică cu obiectele bazei de date Oracle (tabele, vederi, grupuri, instantanee, secvenţe, sinonime, roluri şi spaţii-tabel) prin intermediul instrucţiunilor.

Instrucţiunile SQL sunt directive care specifică serverului Oracle să efectueze o acţiune. Instrucţiunile SQL,formulate conform unei sintaxe specifice încep cu o comanda, urmată de restul instrucţiunii care conţine elemente de bază ale limbajului O instrucţiune SQL incompletă nu poate fi executată de serverul Oracle care returnează o eroare.

Instrucţiunile limbajului SQL pot fi grupate în cinci categorii:

1. **limbaj de interogare (SELECT)** permite regăsirea datelor memorate în tabelă;
2. **limbajul de definire a datelor (DDL** - Data Definiton Language): instrucţiunile CREATE, ALTER, DROP utilizate dinamic, permit definirea, modificarea şi ştergerea structurilor de date, a tabelelor care compun baza de date,
3. **limbaj de manipulare a datelor (DML**-Data Manipulation Language): instrucţiunile: INSERT, UPDATE, DELETE permit inserarea, modificarea şi ştergerea rândurilor tabelei.
4. **limbaj de control al datelor (DCL** -Data Control Language)): instrucţiunile GRANT, REVOKE utilizate pentru definirea şi modificarea drepturilor utilizatorilor;
5. **comenzi de control al tranzacţiilor (TC** -Transaction Control);

Numele obiectelor bazei de date

Obiectele bazei de date: tabele, coloanele, vederi, grupuri, indecşi, secvenţe, sinonime, roluri au nume. Limbajul SQL impune regulile următoare pentru denumirea obiectelor, precum şi a utilizatorilor

- Numele trebuie să aibă între 1 si 30 de octeţi (pe un octet se scrie un caracter)
- Numele trebuie să înceapă cu un caracter alfabetic (o literă)
- Numele pot conţine numai caractere alfanumerice şi caracterele _, $ şi #.
- Numele trebuie să fie unic în propriul spaţiu al numelor.
- Nu se face distincţie între majuscule şi litere mici.
- Un nume poate fi încadrat între ghilimele duble

Cuvintele rezervate ale limbajului SQL este un limbaj alcătuit din cuvinte cheie, rezervate. Ex: ALTER, CHECK, DESCRIBE, HAVING, WHERE, LIKE, SUM, TABLE

Constantele sunt valori fixe, care nu se modifică în timp

- Constante numerice (valori de tip numeric) exemple: 3, 4, 45, .89

- Constante alfanumerice (şir de caractere) exemple: ’program’, ’operator’

Variabilele sunt date care pot avea valori diferite în timp

- Variabile sistem
- Variabile asociate numelui coloanelor

Operatori sunt simboluri grafice/cuvinte rezervate care definesc o operaţie logică sau aritmetică (înmulţire *, împărţire /, adunare +, concatenare şi de caractere II). În tabelul 2.1 sunt prezentaţi operatorii limbajului

Tabel 2.1. Operatorii folosiţi în limbajul SQL

Tipul operatorului	Operaţia	Simbol grafic	Exemplu
Aritmetici	Adunare	**+**	
	Scădere	**-**	
	Înmulţire	*****	
	Împărţire	**/**	
Alfanumerici	Concatenare	**I I**	**’pro’II ’gram’=’program’**
De Comparaţie	Verifică dacă un şir de caractere se potriveşte unui model Caracterul**’_’** înlocuieşte orice caracter; Caracterul procent **%** înlocuieşte mai mult de două caractere sau zero	**LIKE**	**’Mama’ LIKE ’_a%’** returnează TRUE **’Olaru’ LIKE ’_a%’** returnează FALSE
	Testează dacă o valoare se găseşte într-un interval de valori	**BETWEEN**	(x>a) and (x<b) echivalent cu: **x BETWEEN a AND b**
	Testează dacă o valoare este într-o mulţime de valori	**IN**	(x=a) OR (x=b) OR (x=c) echivalent cu **x IN (a,b,c)**
	Testează dacă o expresie are valoarea NULL	**IS NULL** **IS NOT NULL**	NULL indică absenţa unei valori într-o celulă a unei coloane şi nu valoarea zero
Logici	Negaţia logică	**NOT**	
	Şi logic	**AND**	a AND b= TRUE numai dacă a şi b sunt TRUE
	Sau logic	**OR**	a OR b= TRUE cel puţin unul dintre a sau b au valoarea TRUE

Expresiile sunt secvenţe formate din constante, variabile, operatori, funcţii. Evaluarea unei expresii se face de la stânga la dreapta respectând ordinea de prioritate a algoritmilor. Rezultatul evaluării este o valoare de tip: numeric, alfanumeric (şir de caractere) sau logic. Ordinea priorităţii operatorilor logici: **NOT, AND, OR**

2.2. Limbajul de interogare a bazelor de date

Instrucţiunea SELECT are ca efect extragerea, afişarea, regăsirea datelor din una au mai multe tabele. Datele returnate de instrucţiune vor fi afişate tabelar (pe linii şi coloane) sau vor popula o altă tabelă .

Forma cea mai simplă a instrucţiunii este:

SELECT <lista_coloane> (lista cu numele coloanelor separate prin virgulă)

FROM <nume_tabel>; (tabelul din care selectăm datele)

Tabel 2.2. Operaţiile realizate de instrucţiunea SELECT

Nr. crt	Tip operaţie	Efectul
1.	Proiecţia	Afişarea datelor din coloanele care îndeplinesc anumite criterii
2.	Selecţie (filtrare)	Afişarea datelor din rândurile care îndeplinesc anumite criterii
3.	Unirea tabelelor	Afişarea datelor din tabele aflate în relaţie

Clauzele instrucţii SELECT sunt scrise într-o ordine bine determinată, şi realizează operaţiile conform tabelului de mai jos:

Tabel 2.3. Ordinea clauzelor SELECT

Clauza	Acţiunea realizată	Descriere	Obligatorie în instrucţiune
SELECT	Regăsirea/afişarea datelor	Afişează coloanele sau valoarea expresiilor	Da
FROM	Indică tabela	Tabela/tabele din care selectăm datele	Da
WHERE	Filtrarea datelor	Filtrare la nivel de rând/ rândurile care îndeplinesc anumite criterii	Nu

GROUP BY	Gruparea datelor	Specificaţie de grup/pentru coloane cu valori identice (un grup)	Numai dacă se calculează şi afişează valorile funcţiilor agregat pentru grupuri
HAVING	Filtrarea grupurilor	Filtrarea grupurilor	Numai afişarea grupurilor care îndeplinesc anumite criterii
ORDER BY	Sortarea datelor	Ordonează datele implicit crescător, şi descrescător cu DESC	Nu

Pentru exemplificarea instrucţiuilo din aces capitol am folosit mdelul concepţional al bazei de date pe care am numit-o „Depozitul de calculatoare" proiectată în capitolul IV

2.2.1 Afişarea datelor din tabele

Tabel 2.4. Instrucţiuni simple SELECT

<table>
<tr><td>Instrucţiunea</td><td>Rezultat</td></tr>
<tr><td colspan="2">1 Afişarea coloanelor individuale
Returnează o singură coloană şi toate rândurile dintr-un tabel</td></tr>
<tr><td>SELECT nume_produs
FROM produse;</td><td><table><tr><th>nume_produs</th></tr><tr><td>mouse</td></tr><tr><td>tastaturi</td></tr><tr><td>memorii</td></tr><tr><td>placa_video</td></tr><tr><td>placa_reţea</td></tr><tr><td>monitor</td></tr><tr><td>microfon</td></tr></table></td></tr>
<tr><td colspan="2">2. Afişarea coloanelor multiple specificate în lista_coloane</td></tr>
<tr><td>SELECT id_produs, nume_produs, preţ
FROM produse;</td><td><table><tr><th>id_produs</th><th>nume_produs</th><th>preţ</th></tr><tr><td>11</td><td>mouse</td><td>12</td></tr><tr><td>12</td><td>tastaturi</td><td>23</td></tr><tr><td>21</td><td>memorii</td><td>34</td></tr><tr><td>22</td><td>placa_video</td><td>14</td></tr><tr><td>31</td><td>placa_reţea</td><td>24</td></tr><tr><td>32</td><td>monitor</td><td>13</td></tr><tr><td>41</td><td>microfon</td><td>21</td></tr></table></td></tr>
<tr><td colspan="2">3.Afişarea tuturor coloanelor. Caracterul de înlocuire * este plasat în locul numelor coloanelor</td></tr>
</table>

SELECT *

FROM producători;

id_producător	nume_producator	localitate
1	Polirom	Berlin
2	Minerva	Sofia
3	Donaris	Budapesta
4	Corint	Londra

SELECT DISTINCT localitate

FROM clienţi

Bucureşti
Ploieşti
Braşov
Sibiu

2.2.2. Ordonarea datelor regăsite Clauza ORDER BY

Sortarea (ordonarea) implicit crescătoare a datelor de ieşire se realizează prin folosirea instrucţiunii ORDER BY.

SELECT <nume_coloane>

FROM <nume_tabela>

ORDER BY<criteriu_ordonare>; un nume de coloană sau o expresie

- Pentru a sorta în ordine descrescătoare, este necesară specificarea cuvântului chei DESC imediat după fiecare coloană.
- În clauza ORDER BY se poate specifica poziţia relativă a coloanei selectate din SELECT. Exemplu: ORDER BY 2 va indica sortarea după coloana 2 în SELECT
- Ordonarea se poate realiza după una sau două coloane; numele lor este specificat în clauza ORDER BY;

Tabel 2.5. Instrucţiuni SELECT -ORDER BY

Instrucţiunea	Rezultat
1 Sortarea după o coloană Returnează coloana nume_produs în care datele sunt ordonate crescător alfabetic, valorile NULL apar la sfârşit	
SELECT nume_produs FROM produse ORDER BY nume_produse**;**	nume_produs memorii microfon monitor mouse placa_reţea placa_video tastaturi
Ordonarea descrescătoare numai după preţ	

SELECT id_produs, nume_produs, preţ FROM produse ORDER BY preţ DESC, nume produs;	id_produs | nume_produs | preţ 42 | microfon | 80 21 | memorii | 34 31 | placa_reţea | 24 12 | tastaturi | 23 41 | microfon | 21 22 | placa_video | 14 32 | monitor | 13 11 | mouse | 12

2.2.3. Câmpurilor cu valoare calculată Alias-ul unei coloane

Un câmp cu valoare calculată este o colană care nu există efectiv în tabela din baza de date dar este creată de instantaneu de instrucţiunea SELECT şi afişează valori calculate ca rezultat al rezolvării unor expresii: afişarea modificării unui preţ, afişarea valorii totale a fiecărui articol din tabelă (produsul dintre preţ şi cantitatea), afişarea numelui concatenat cu prenumelui persoanelor

Baza de date „recunoaşte” care dintre coloanele incluse într-o instrucţiune SELECT reprezintă coloanele reale ale unui tabel şi care sunt câmpurile calculate. Pentru a da un nume câmpurilor calculate (ALIAS) se foloseşte **clauza AS** urmată de numele atribuit câmpului calculat. Numele este afişat cu majuscule.

Alias-ul câmpului calculat scris într-un anumit format, între ghilimele, este afişat în acelaşi format.

Tabel 2.6. Crearea câmpurilor calculate

Instrucţiunea	Rezultat
Câmpul calculat ”titlul_producător” format prin concatenarea şirurilor de caractere din coloana „numele” şi coloana „localitate”; II-operator concatenare	
SELECT numele II localitate AS ”titlul_producător” FROM producători ORDER BY numele;	titlul_producător Corint Londra Donaris Budapesta Minerva Sofia Polirom Berlin

SELECT id_produs, preţ, cantitate,
cantitate*preţ AS ”Valoare_totală”
FROM produse
WHERE id_producator=2
ORDER BY preţ;

id_produs	preţ	cantitate	valoare _totală
22	14	3	42
21	34	5	170
23	80	2	160

2.2.4. Filtrarea datelor (selecţia) Clauza WHERE

Selecţia (filtrarea) şi afişarea rândurilor care îndeplinesc un criteriu_filtrare se realizează prin folosirea clauzei WHERE.

SELECT <nume_coloane>

FROM <nume_tabela>

WHERE<criteriu_filtrare>

Criteriul de filtrare este o expresie care precizează condiţiile pe care trebuie să le îndeplinească o linie pentru a fi afişată. La construirea criteriului se folosesc operatorii specificaţi în Tabel 2/1 Operatorii folosiţi în limbajul SQL.

Tabel 2.7. Filtrarea datelor

Afişează numele clienţilor **numai** din localitatea Bucureşti

SELECT numele, localitate
FROM clienţi
WHERE localitate=’Bucureşti’**;**

numele	localitate
Nicolae	Bucureşti
Tatu	Bucureşti

Afişează numele şi localitate numai pentru clienţii al căror nume includ caracterul ’a’

SELECT numele, localitate
FROM clienţi
WHERE numele LIKE %’a’%

numele	localitate
Nicolae	Bucureşti
Stanciu	Braşov
Tatu	Bucureşti
Stan	Ploieşti

SELECT id_producător, nume_produs
FROM produse
WHERE id_producător<>2;

id_producător	nume_produs
1	mouse
1	tastaturi
3	placa_reţea
3	monitor
4	microfon

Verifică încadrarea într-un interval de valori : preţul să fie între 25 şi 50

SELECT nume_produs, preţ
FROM produse
WHERE preţ BETWEEN 25 and 50 **;**

nume_produs	preţ
memorii	34
microfon	80

Filtrarea avansată datelor

Pentru a crea condiţii de căutare puternice şi a asigura un control mai strict a operaţiei de filtrare, SQL permite specificare mai multor clauze în WHERE. Aceste clauze se pot folosi în două moduri: sub forma de clauze AND şi clauze OR.

Tabel 2.8. Filtrarea avansată datelor

Utilizarea operatorului AND	Rezultatul instrucţiunii
SELECT id_produs, preţ, nume_produs FROM produse WHERE id_producător=2 AND preţ>20	Afişează numai produsele cu preţ>20 realizate de un anumit producător (id_producător=2).
Utilizarea operatorului OR	
SELECT nume_produs, pret FROM produse WHERE id_producător=2 OR id_producător=3	Afişează numele şi preţul produselor realizate de unul sau altul dintre cei doi producători. Se afişează toate rândurile care corespund uneia dintre condiţiile specificate
SQL prelucrează operatorii AND înainte operatorilor OR Pentru a grupa operatorii în mod explicit se folosesc parantezele care au prioritate mai mare decât AND	

SELECT nume_produs, preţ FROM produse WHERE (id_producător=2 OR id_producător=3) AND pret>50;	Afişează numele produsului cu preţul mai mare decât 50 livrat de unul dintre producătorii cu id-urile menţionat în paranteze (numai producătorul cu id_producător are un produs care satisface această clauză)
Operatorul IN se foloseşte pentru selecţia dintr-un domeniu de valori	
SELECT nume_produs, pret FROM produse WHERE id_producător **IN (1,2,4)** ORDER BY nume_produs;	Afişează numele şi preţul produselor realizate de producătorii ale căror id_producător au valoarea în lista de valorilor Operatorul IN are acelaşi rol ca OR. Dar se foloseşte IN când se lucrează cu liste lungi şi ordinea de evaluare este mai uşor de controlat la operatorul IN
Operatorul NOT se utilizează în **conjuncţie cu operatorul IN** pentru a afişa toate rândurile care nu corespund listei de criterii	
SELECT nume_produs, FROM produse WHERE NOT id_producător IN (1,2,4) ORDER BY nume_produs	Afişează numele produselor numai pentru producătorii al căror id_producător nu se regăsesc în listă de valori Instrucţiunea care are o acţiune similară: WHERE (id_prodcător<>1 OR id_prodcător<>2 OR id_prodcător<>1)

2.2.5. Utilizarea funcţiilor singulare de manipulare a datelor

Funcţiile sunt operaţii care se execută cu datele din bazele de date în vederea facilitării conversiei şi manipulării datelor

Tabel 2.9. Clasificare funcţiilor după numărul de rânduri pe care acţionează

Categoria	Se aplică pentru	Folosite în clauza	Funcţii pentru tipul date
Funcţii singulare	O înregistrăre **un rând**	SELECT realizarea de calcule WHERE selecţia rândurilor	▪ **Şir de caractere** ▪ **Date numerice** ▪ **Date calendaristice**

			▪ **De conversie**
Funcţii de grup	**grup de înregistrări**	SELECT	▪ **Numerice**

Funcţiilor singulare se clasifică după tipul de date asupra cărora operează :

- Funcţii singulare de manipulare a textului (Tabel 2.10.);
- Funcţii singulare aplicate datelor numerice (Tabel 2.11.);
- Funcţii singulare aplicate datelor calendaristice (Tabel 2.12.);
- Funcţii singulare de conversie (Tabel 2.13.);
- Funcţii singulare de uz general (Tabel 2.14.);

Tabel 2.10. Funcţii singulare de manipulare a textului (pentru şiruri de caractere)

Funcţia	Descriere
LOWER(şir)	Converteşte şirul în minuscule
UPPER(şir)	Converteşte toate caracterele şirul în majuscula
Funcţia	Descriere
INITCAP(şir)	Converteşte la majuscul primul caracter din fiecare cuvânt, restul caracterelor sunt scrise cu minuscule
CONCAT (şir1, şir2)	Concatenează două şiruri de caractere
SUBSTR(şir, poz, nr)	Extrage din şir, începând cu poziţia poz, de ;la stânga, un număr de nr. caractere; dacă poz este minus se începe extragere din dreapta
LENGHT(sir)	Returnează numărul de caractere din şir (lungimea şirului)
LPAD (şir1, nr, şir2)	Completează la stânga şir1 cu caractere din şir2 până şir1 are nr caractere
RPAD(şir, nr, şir2)	Completează la dreapta şir1 cu caractere din şir2 până şir1 are nr caractere
RTRIM(şir)	Elimină spaţiile goale de la dreapta şirului
REPLACE(şir1, şir2, şir_nou)	Înlocuieşte toate apariţiile şir2 din şir1 cu sir_nou

Tabel 2.11. Funcţii aplicate datelor numerice

Nume_funcţie(x) (argumentul este o dată	Descriere

numerică)	
ABS(-4.5)	Returnează *valoarea absolută* adică 4.5
POWER(2,4)	Calculează şi Returnează 2 la *puterea* 4 adică 16
SQRT(4)	Calculează şi returnează *rădăcina pătrată* a argumentului rezultatul este 16
MOD(7,2)	returnează *restul întreg al împărţirii* lui 8 la 3 conform teoremei împărţirii cu rest 7=3*2+1 adică returnează restul 1
ROUND(x,y) rotunjeşte valoarea lui x la un număr de cifre precizat prin y	Round(745.124,2)= 745.12 Round(745.126,2)=745.13 Round (745.126,0)=745 Round(745.897)=746 Round(745.126,-2)=700 Round(745.126,-3)=0

Tabel 2.12. Funcţii aplicate datelor calendaristice

SYSDATE	Returnează data şi ora serverului bazei de date
CURRENT_DATE	Returnează data şi ora curentă a aplicaţiei client care poate să difer de aplicaţa server
SYSTIMESTAMP	Returnează data în format TIMESTAMP
ADD_MONTHS(data, nr_luni)	Adaugă /scade un nr de luni(-nr_luni) la data curentă
MONTHS_BETWEEN (data1, data2)	Calculează şi afişează nr de luni dintre două date calendaristice
LAST(data1, data2, data3…)	Afişează cea mai veche dată din cele din listă
GREATEST(data1, data2, data3…)	Afişează cea mai recentă dată din cele din listă
LAST_DAY(data)	Returnează ultima zi din luna din care face parte data
NEXT(data, monday)/NEXT(data, Tuesday)	Returnează următoarea dată imediat după Monday şi Tuesday
ROUND(data, 'format')	Rotunjeşte data transmisă la cea mai apropiată oră dacă ora este după12 AM se returnează data

	următoare după data din argiment

Tabel 2.13. Funcţii de conversie

TO_CHAR(dt,format)	Converteşte data calendaristică dt în şirul de caractere precizat de formar
TO_DATE(şir, format)	Converteşte şirul de caractere în dată calendaristică
TO_CHAR(nr,format)	Converteşte un nr în şir de caractere
TO_NUMBER(şir, format)	Converteşte şir de caractere în valoare numerică

Talel 2.14. Funcţii de uz general

NVL(val1,val2)	Returnează numai valoarea care nu este nulă
NVL2(val1, val2, val3)	Dacă val1 este nulă returnează val2 şi dacă nu este NULL returnează val3
NULLIF(expr1, expr2)	Dacă expr sunt egasle se returnează NULL altfel se returnează prima expresie
DECODE	Dacă val1 este nulă returnează val2 şi dacă nu este NULL returnează val3

2.5.6. Funcţiile de grup (funcţii agregat)

Sunt utilizate pentru obţinerea datelor statistice (informaţii de sinteză) .

Returnează valorile maxime, minime, medii, contorizări, sume, produs, numărul de înregistrări care îndeplinesc o anumită condiţie. Tipul valorii returnate este numeric

Tabel 2..15. Funcţiile returnează o singură valoare

Exemplu	Descrierea acţiunii	Rezultat
AVG() Returnează media valorilor dintr-o coloană (NULL este ignorat)		
SELECT AVG(preţ) AS preţ_mediu FROM produse;	Afişează preţul mediu al produselor care au o valoare introdusă în coloana preţ	Preţ_mediu ---------- 27.63

SELECT AVG(NVL(preţ,0)) AS preţ_mediu FROM produse;	NVL(preţ,0) înlocuieşte cu zero valorile de NULL Calculează media numai pentru rândurile în care nu sunt valori NULL)	

Funcţiile de grup (funcţii agregat)

COUNT() Afişează numărul rândurilor dintr-o coloană care nu conţin NULL		
SELECT COUNT(nume_produs) AS nr_prod FROM produse WHERE id_producător=2 Afişează numărul de produse livrate de producătorul cu id_producator=2 SELECT COUNT(*) AS nr_produse FROM produse	Returnează numărul rândurilor care conţin valori din coloana dată ca parametru; COUNT(*) returnează numărul de înregistrări dintr-o tabelă care conţin valori ,Valorile NULL sunt ignorate	nr_prod ---------- 3
MAX() returnează valoarea maximă dintr-o coloană		
SELECT MAX(nume_produs) AS preţ_max FROM produse WHERE id_producător=2 Produsul cu preţul cel mai mate livrat de producătorul specificate în clauza WHERE	Returnează cea mai mare valoare dintr-o coloană specificată (Valoarea NULL este ignorată)	preţ_max ---------- 80
MIN() returnează valoarea minimă dintr-o coloană		
SELECT MIN(nume_produs) AS preţ_min FROM produse WHERE id_producător=2 Ce se afişeză dacă lipseşte clauza WHERE?	Returnează preţul minim al produselor livrate de producătorul specificat in clauza WHERE (valorile NULL sunt ignorate)	preţ_min ---------- 14

SUM() returnează suma valorilor incluse în cloană		
SELECT SUM(cantitate) AS stoc FROM produse SELECT SUM(preţ*cantitate) AS valoare_totală FROM produse WHERE id_producator=2	Returnează suma (totalul) valorilor dintr-o coloană SUM(cantitate) Returnează suma unui câmp calculat (valoarea totală) SUM(preţ*cantitate)	stoc ---------- 35

Modalităţi de folosire a funcţiilor agregat

- **Funcţii agregat cu valori distincte**

Efectuarea de calcule cu toate rândurile. (ALL este o valoare prestabilită şi nu se specifică)

Afişează numărul produselor distincte din tabela PRODUSE indiferent de producător

```
SELECT COUNT (DISTINCT id_produs) AS nr_produse
FROM produse;
```

Efectuarea de calculelor numai cu valorile distincte(se specifică clauza **DISTINCT)**

Afişează preţul mediu numai pentru produsele distincte, livrate de un anumit producător

```
SELECT AVG(DISTINCT preţ) as preţ_mediu
FROM produse
WHERE id_producător=2
```

- **Combinarea funcţiilor agregat**

Într-o instrucţiunile SELECT se pot folosi mai multe funcţii agregat:

```
SELECT  COUNT(*) AS ”nr_articole”
        MIN(preţ) AS ”preţ_minim”
        MAX(preţ)  AS ”preţ_maxim”
        AVG(preţ) AS ”preţ_medi”u
FROM produse;
```

Rezultatul este:

```
nr_articole     preţ_minim    preţ_maxim    preţ_mediu
-------------------------------------------------------------------
8               12            80            27.63
```

Funcţiile de grup nu se folosesc în clauza WHERE. Pentru a afişa produsele cu preţul mai mare decât preţul mediu se folosesc sub interogările (vezi sub interogări)

2.2.7. Gruparea datelor şi selectarea grupurilor GROUP BY şi HAVING BY

Cele două clauze se folosesc pentru a realiza calcule statistice pe grupuri de date şi a afişa rezultatele pentru fiecare grup în parte (gruparea datelor şi selecţia grupurilor)

- **Clauza GROUP BY<criteriul_grupare>** grupează datele în funcţie de criteriul_grupare, iar pentru fiecare grup în parte se aplică funcţia agregat specificată în SELECT . Criteriul_grupare este numele unei coloane

Tabel 2.15. Gruparea datelor

Instrucţiunea	Rezultat
SELECT id_producător, COUNT(*) AS ”nr_produse” FROM produse GROUP BY id_producător; Afişează numărul de produse livrat de fiecare producător	id_producător nr_produse ---------- 1 2 2 3 3 2 4 1

În clauza GROUP BY

- nu se acceptă aliasul coloanei;
- sunt enumerate toate coloanele care apar şi în SELECT în afara funcţiilor de grup
- pot să apară şi alte coloane care nu sunt în SELECT
- sortarea grupurilor se realizează cu clauza ORDER BY clauza de ordonare este aceiaşi cu cea de grupare

Exemplu: SELECT id_producător, MAX(SUM(id_produs)) AS ”max_produse”
FROM produse
GROUP BY id_producător

această declaraţie afişează numai id_producător care livrează cele mai multe produse

- **Clauza HAVING <criteriu_de_selecţie>** Selectează grupurilor care îndeplinesc criteriul_de _selecţie reprezentat printr-o expresie logică în care se testează valorea returnată de funcţia agregat din SELECT Tabel 2/16

Comparaţie între Clauzele WHERE şi HAHING

WHERE Selectează toate rândurile care îndeplinesc o condiţie

HAVING Selectează toate grupurile care îndeplinesc o condiţie

Toate tehnicile folosite la WHERE se pot aplic şi al HAVING

Tabel 2.16. Exemple de folosire a clauzele WHERE şi HAHING

Declaraţia instrucţiunii	Rezultat
SELECT id_producător, COUNT (*) AS nr_produse FROM produse WHERE preţ>20 GROUP BY id_producător HAVING COUNT(*)>2	Afişează producătorii care oferă mai mult de două produse la un preţ mai mare de 20

Explicaţia exemplului din tabelul 2.16.

- SELECT proiectează două coloane: id_producător nr_produse
- FORM indică tabela din care se prelucrează datele;
- WHERE selectează rândurile care îndeplinesc condiţia preţ>20
- GROUP BY grupează în funcţie de id_producător rândurile selectate de WHERE
- Funcţia agregat COUNT(*) va număr produsele pentru fiecare grup
- HAVING selectează şi permite afişarea producătorulor (id_producător) care au mai mult de 2 produse cu preţul >20

2.2.7. Subinterogări sunt instrucţiuni SQL imbricate: o interogare inclusă în altă interogare, o interogare interioară şi o interogare exterioară şi sunt folosite atunci când dorim să afişăm informaţii dintr-o tabelă pe baza informaţiilor din aceiaşi tabelă sau din tabele diferite

Tabel :2.17 Tipuri de interogări

Tipul	Tipul valorii returnat	Unde apar	Operatorii folosiţi

Simplă	Returnează o singură valoare	WHERE şi HAVING	Comparaţie <,=,>
Multiplă	Returnează o mulţime de valori	WHERE	IN, ALL ANY

Reguli de lucru cu sub interogările

- se execută mai întâi sug interogările, incluse între paranteze;
- sub interogarea nu conţine clauza ORDER BY;

Studiu de caz: Pentru tabelele COMENZI şi ARTRICOL

Tabel 2.18 Sub interogare multiplă (returnează o mulţime de valori)

Afişeze toţi clienţii (id_client) care au comandat produsul cu id_produs=31	SELECT id_client FROM comenzi WHERE nr_comandă IN (SELECT nr_comanda FROM articol WHERE id_produs= 31)
	SELECT id_client FROM comenzi WHERE nr_comandă IN (11,24)
	Rezultatul va fi: id_client ---------- 1 2

Tabel 2.20. Sub interogare simplă (returnează o singură valoare)

Explicaţie: Afişeze clienţii (id_client) care au comandat produsul cu id_produs=31	SELECT nume_produs, preţ FROM produse WHERE preţ> (SELECT AVG(preţ) FROM produse)
	SELECT nume_produs, preţ FROM produse WHERE preţ>27,63
	Rezultatul va fi nume_produs preţ ---------- memorii 80 microfon 34

2.2.9. Interogări multiple Unirea tabelelor (JOIN)

Capacitatea de unire instantanee a tabelelor în cadrul interogărilor de regăsire a datelor este una dintre cele mi puternice facilităţi ale limbajului SQL..

O uniune (JOIN) este un mecanism folosit pentru asocierea tabelelor în cadrul unei instrucţiuni SELECT, folosind o sintaxă specială astfel încât să se afişeze un set de dat de ieşire . Operaţia JOIN se realizează în clauza WHERE

Tabel 2.21. Tipuri de operaţii JOIN

Tip de JOIN	Rezultatul
PRODUSUL CARTEZIAN	Legă fiecare înregistrare dintr-o tabelă1 cu toate înregistrările din tabela2
EQUIJOIN	Leagă două tabele care au o coloană identică (condiţie de egalitate)
NONEQUIJOIN	Leagă două tabele folosind un alt operator decât cel de egalitate
SELF JOIN	Uniuni interioare unei tabele se foloseşte în relaţiile recursive
OUTERJOIN	Se foloseşte atunci când vrem să afişăm înregistrările dintr-o tabelă pentru care nu există corespondent în cealaltă da tabelă

Tabel 2.22. Exemple de aplicare a tipurilor operaţii JOIN

1. **Produsul cartezian**	
SELECT nume_producător,nume_produs, preţ_produs FROM producători, produse;	Datele de ieşire reprezintă combinaţii între fiecare producător , fiecare articol,
2. **EchiJOIN**	
SELECT a.nr_comanda, a.cantitate, p.nume_produs FROM articol a, produse a WHERE a.id_produs=p.id_produs	Afişează nr_comandă, numele produsului comandat şi cantitatea comandată (a,p sunt alias-uri tabele)
3. **NonechiJOIN**	
SELECT p.nume_produs, a.cantitate, FROM produse p, articol a WHERE p.cantitate<a.cantitate	Afişează numele produselor pentru care s-a comandat o cantitate mai mare decât cea existentă în tabela produse

4. **SelfJOIN**	
SELECT a. id_client, a.numele, a.telefon, FROM clienţi a,clienti b WHERE a. job_client=b.job_client AND b.job_client=informatician	Pentru a transmite un mesaj tuturor clienţilor care au job=informatician
5. **OuterJOIN** (**LEFT**) sunt afişate toate înregistrările din tabela produse cu sau fără corespondent din tabela comenzi **OuterJOIN** (**RIGHT**) sunt afişate toate înregistrările din tabela comenzi cu sau fără corespondent din tabela produse	
SELECT p.nume_produs, c. nr_comandă FROM produse p, comenzi c WHERE p. id_produs=c.id_comenzi (+) (LEFT) SELECT c. nr_comandă p.nume_produs FROM comenzi produse p, WHERE c.id_comenzi (+)=p. id_produs; (RIGHT)	Afişează toate produsele cu sau fără comandă

2.2.10 Interogări compuse Combinarea interogărilor

Interogările SQL se combină folosind operatorul **UNION**, iar rezultatele afişate reprezintă o combinaţie a unor seturi de date

Studiu de caz: Realizarea unui raport care să conţină toţi clienţii din oraşele Bucureşti şi Ploieşti care au numele ’Ionescu’

Rezolvare se realizează uniunea a două instrucţiuni SELECT (Tabelul 2.23.)

Tabel 2.23 Aplicarea instrucţiunii UNION .

Instrucţiuni	Rezultat
SELECT numele, telefon FROM clienţi WHERE localitate IN (’Bucureşti’,’Ploieşti’) **UNION**	numele | telefon Nicolae | 2345 Ionescu | 3457 Stanciu | 4567 Ionescu | 7890 Stan | 3456

SELECT numele, telefon FROM clienţi WHERE numele=’Ionescu’	<table><tr><th>numele</th><th>telefon</th></tr><tr><td>Ionescu</td><td>3457</td></tr><tr><td>Ionescu</td><td>7890</td></tr></table>

Tabel 2.24. Operatorii de combinarea rezultatelor a două sau mai multor interogări sunt:

Operatorul	Acţiunea realizată
UNION ALL	Afişează toate rândurile rezultate din interogările pe care le leagă, inclusiv duplicatele
UNION	Afişează toate rândurile rezultate din interogările pe care le leagă, elimină duplicatele
INTERSECT	Afişează liniile returnate de ambele interogări
MINUS	Afişează liniile care sunt returnate de prima interogare dar nu sunt returnate de a doua

2.3. Limbajul de definire a datelor (DDL)

2.3.1 Crearea tabelelor

1. Instrucţiunea **CREATE TABLE** se utilizează pentru crearea structurii unui tabel nou.

Forma generală a instrucţiunii este:

CREATE TABLE <nume_tabel>

(

<nume_coloană1><tipul de dată><dimensiunea>

<nume_coloană2><tipul de dată><dimensiunea>

..);

În tabelul 2.25. sunt prezentate instrucţiuni de creare a structurii tabelelor pentru baza de date „Depozitul de calculatoare”

Tabel 2.25 Exemple de aplicare a instrucţiunii CREATE TABLE

Instrucţiune	Analiza/Explicaţii
CREATE TABLE Produse	- numele tabelului este unic în baza de date şi

(id_produs number (3), id_producător number (3), denumire varchar2 (20), preţ number (6,2), descriere clob,);	este scris imediat după cuvintele cheie CREATE TABLE - numele unei coloane este unic în cadrul tabelului, urmat de tipul de date al coloanei şi dimensiunea; - coloanele sunt separate unele de altele prin virgulă
CREATE TABLE Producători (id_producător number (3), nume_producător varchar2 (20), localitate varchar2 (20), telefon number (13));	instrucţiune creează tabelul Producători
Clauza DEFAULT specifică valorilor prestabilite pentru o coloană	
CREATE TABLE Clienţi (id_client number (13), numele varchar2 (30), localitate varchar2 (20) DEFAULT ’Bucureşti ’’ localitate varchar2 (20), telefon number (13)); data_ang date DEFAULT SYSDATE	Valorile prestabilite pentru o coloană sunt precizate folosind cuvântul cheie DEFAULT în definiţia coloanelor În clauză se precizează valoarea prestabilită Valorile DEFAULT se folosesc în coloanele care se vor utiliza pentru calcule sau gruparea datelor

2. **Crearea unei tabele folosind un tabel existent** se folosesc sub interogările

Exemplu: CREATE TABLE bursieri

AS (SELECT id_elev, nume, prenume

FROM elevi

WHERE BURSIER=’D’)

Subinterogarea selectează din tabela ELEVI numai elevii care îndeplinesc condiţia din clauza WHERE care vor popula tabela BURSIERI (coloanele id_elev, nume, prenume)

2.3.2 Definirea constrângerilor regulilor de integritate a tabelelor

Limbajului SQL foloseşte pentru reprezentarea regulilor de integritate, constrângerile definite prin cuvinte cheie. Constrângerile garantează că datele introduse în baza de date sunt corecte şi valide Tipurile de constrângere se definesc la crearea tabelelor şi sunt cele prezentate în tabelul 2.26. şi 2.27.

Tabel 2.26. Tipuri de Constrângeri:

Integritatea	Tip de constrângere	tip_constrângere	Nivelul de definire
Entităţilor	De integritate a tabelei	**PRIMARY KEY**	Tabel/coloană
Referenţială	De integritate referenţială	**FOREIGN KEY**	Tabel/coloană
De domeniu	De domeniu	**NOT NULL**	Coloană
		UNIQUE	Tabel/coloană
		CHECK	Tabel/coloană

Tabel 2.27. Definirea constrângerilor

La nivel de coloană	<nume_coloană><tip_dată><tip_constrângere>
	<nume_coloană><tip_dată> <CONSTRAINT <nume_constrângere> <tip_constrângere>;
La nivel de tabelă	<tip_constrângere>
	CONSTRAINT <nume_constrângere> <tip_constrângere>;

Numele constrângerii se formează din numele tabelei şi prefixele numelui restricţiei

Tabel 2.28. Pprefixele constrângerilor

Nume_constrângere	Tip_constângere	Explicaţie

<nume_tabel>_pk	PRIMARY KEY	pentru cheile primare
<nume_tabel>_un	UNIQUE	pentru constrângeri de unicitate
<nume_tabel>_nn	NOU NULL	pentru atributele obligatorii
<nume_tabel>_ck	CHECK	pentru reguli de validare la nivel de atribut sau înregistrare
<nume_tabel>_fk	FIREIGN KEY	pentru cheie străină

Tipurilor de constrângeri

1. Cheie primară (PRIMARY KEY)

Cheia primară este o coloană sau o combinaţie de coloane care identifică în mod unic fiecare înregistrare dintr-un tabel, în care valorile sunt unice şi nenule

Cheia primară impune integritatea la nivel de tabel şi facilitează stabilirea relaţiilor cu alte tabele din baza de date. De regulă, cheile primare vor fi nişte coduri numerice, uneori generate chiar de sistemul de gestiune a bazei de date

Cheie primară simplă definită pentru un singur câmp se defineşte la nivel de coloană.

Cheie primară compusă din două sau mai multe câmpuri se defineşte la nivel de tabelă.

O coloană dintr-un tabel declarată cheie primară satisface următoarele condiţii

- condiţie de unicitate valorile coloanei cheie primară sunt unice în toate înregistrările Fiecare rând trebuie să aibă o valoare a cheii primare
- condiţia NOT NULL nu sunt permise valori NULL
- Coloanele cheii primare nu pot suferii modificări sau actualizări
- Valorile de tip cheie primară nu pot fi refolosite. Dacă un rând este şters din tabel, cheia sa primară nu trebuie atribuită altor rânduri

Tabel 2/28 Definirea cheii primare la nivel de coloană (simplă)

CREATE TABLE producători (id_producător number (13) **PRIMARY KEY** nume_producător varchar2 (20) localitate varchar2 (20), telefon number (13));	CREATE TABLE producători (cnp number (13) CONSTRAINT angajaţi_pk **PRIMARY KEY** nume_producător varchar2 (20) localitate varchar2 (20), telefon number (13));

Tabel 2.29. Definirea cheii primare la nivel de tabel (cheia primară compusa)

CREATE TABLE **angajaţi** (cnp number (13) nr_matricol number (4) nume varchar2 (20) data_naşterii date …….) **PRIMARY KEY(cnp, nr_matricol)**	CREATE TABLE **angajaţi** (cnp number (13) nr_matricol number (4) nume varchar2 (20) data_naşterii date …….) CONSTRAINT angajaţi_pk **PRIMARY KEY(cnp, nr_matricol)**

2. Cheia străina/externă (FOREIGN KEY)

O cheie străină/externă are rolul de a asigura legătura cu alt tabel şi este reprezentată de o coloană dintr-un tabel (copil) ale cărei valori trebuie să corespundă unei coloane cheie primară dintr-un alt tabel (tabelul părinte). Cheile externe constituie o componentă extrem de importantă a asigurării integrităţii la nivel de relaţie (integritate referenţială, la nivel de relaţie).

Stabiliţi relaţia între două tabele prin preluarea unei copie a cheii primare din primul tabel şi introducerea în structura celui de-al doilea tabel, unde devine cheie externă din acest motiv câmpurile cheie primară şi cheie străină au acelaşi tip de date..

Cheia străină contribuie la:

- facilitatea stabilirii relaţiilor dintre perechi de tabele,
- la implementarea şi asigurarea integrităţii la nivel de relaţie; înregistrările din ambele tabele vor fi întotdeauna corelate în mod adecvat, deoarece valorile unei chei externe trebuie să fie identice cu valorile existente ale cheii primare la care face referire.
- evitarea periculoaselor înregistrări „orfane”, un exemplu clasic în acest sens reprezentându-l înregistrarea unei comenzi fără un client asociat. Dacă nu ştiţi cine a emis comanda, nu o puteţi prelucra şi, evident, nu o puteţi factura, iar asta o să vă ducă de râpă vânzările trimestriale

Studiu de caz:

Definirea cheie străine la nivel de coloană:

a) CREATE TABLE Comenzi

(nr_comandă number (5) **PRIMARY KEY,**

id_client number (3) REFERANCE Clienţi(id_client),

data date);

b) CREATE TABLE Comenzi

(nr_comandă number (5) **PRIMARY KEY**

id_client number (3) CONSTRAINT ech_fk REFERANCE Clienţi (id_client),

data date);

Explicaţie: Coloana id_client din tabelul Comenzi acceptă numai valori care se găsesc în coloana (cheie primară) id_client din tabelul Clienţi

Tabelul Comenzi conţine un rând pentru fiecare comandă i iar tabelul Clienţi, informaţiile despre fiecare client Un client poate să facă mai multe comenzi. O comandă este făcută de un singur client. Valorile din coloana id_client din tabelul Comenzi nu sunt în mod necesar unice. Dacă un client a emis mai multe comenzi vor exista mai multe rânduri cu acelaşi identificator de client, deşi numărul comenzii este diferit..

Clauze suplimentare la definirea unei chei străine

ON DELETE CASCADE dacă această clauză este activată atunci la ştergerea unei linii (înregistrări) din tabela părinte (clienţi) se vor şterge toate liniile din tabela copil (comenzi) care au în coloana cheii străine valori din coloana cheie primară a tabelei părinte . (Dacă ştergerea în cascadă este activată şi se şterge un client atunci toate comenzile asociate clientului vor fi şterse în mod automat în tabele comenzi)

ON DELELETE SET NULL dacă această clauza este activată atunci la ştergerea unei înregistrări din tabela părinte, toata înregistrările din tabela copil care fac referire la linia ştearsă (au în câmpul cheie străină valoarea cheii primare) vor primii valoarea NULL.

Dacă nu este precizată una dintre cele două clauze, atunci Oracle nu va permite ştergerea unei înregistrări din tabela părinte, afişarea unei casete care va atenţiona utilizatorul că nu este permisă ştergerea înregistrării indicate.

3. Constrângeri de unicitate cheie unică UNIQUE

Constrângerile de unicitate se folosesc pentru a garanta faptul că toate datele dintr-o coloană (sau dintr-un set de coloane) sunt unice pentru toate înregistrările dintr-o tabelă .

Într-o coloană pentru care se defineşte constrângerea UNIQUE nu pot exista date duplicate. Constrângerea UNIQUE este asemănătoare unei PRIMARY KEY, dar există câteva diferenţe importante:

- Coloanele pot conţine valori NULL, iar aceste valori NULL pot fi oricâte, valoarea NULL este singura valoare care poate fi duplicată într-o coloană UNIQUE.
- Coloanele pot fi actualizate sau modificate.
- Coloanele nu pot fi utilizate pentru definirea cheii primare,
- Valorile din coloanele care prezintă constrângeri de unicitate pot fi reutilizate
- Un tabel poate conţine mai multe constrângeri de unicitate, dar are doar o singură cheie primară

Exemplu:

```
CREATE TABLE ANGAJAŢI
      ( id_ang number(3) PRIMARY KEY,
        cnp  number(13) UNIQUE
        numele  varchar2 (30),
        prenumele varchar2 (30).
        data_ang  date);
```

Explicaţia exemplului: În tabelul **ANGAJAŢI,** fiecare angajat are un cod numeric personal unic, dar pe care nu doriţi să-l folosiţi drept cheie primară pentru că este prea lung (pe lângă faptul că nu doriţi să puneţi la dispoziţie aceste informaţii). Ca atare, fiecărui angajat i se asociază un identificator unic (cheie primară) şi o constrângere de tip UNIQUE asupra coloanei care conţine cnp(codul numeric personal)

4. Constrângerea NOT NULL

NULL este o valoare specială. O celulă primeşte automat NULL dacă nu introduceţi o valoare, ceea ce semnifică faptul că valoarea este nedefinită (nu este zero).

Explicaţie: Într-un EDR, un atribut obligatoriu este marcat cu o steluţă (*), într-o tabelă coloana (câmpul) corespunzător atributului obligatoriu trebuie definită cu constrângerea **NOT NULL** pentru a vă asigura că utilizatorul a introdus o valoare în câmpul respectiv.

- NOT NULL se foloseşte pentru a asigura introducerea unei valori într-un câmp
- NOT NULL se defineşte doar la nivel de coloană.
- NOT NULL poate fi folosită alături de clauza DEFAULT

- Nu confundaţi valorile de NULL cu şiruri vide! O valoare NULL reprezintă lipsa unei valori şi nu un şir vid. Un şir vid este o valoare permisă. Valorile NULL se specifică cu cuvântul NULL şi nu cu un şi vid

Demonstraţie**:** CREATE TABLE ANGAJAŢI

(**id_ang** number(3) PRIMARY KEY,

cnp number(13) UNIQUE

numele varchar2 (30), NOT NULL

prenumele varchar2 (30). NOT NULL

data_ang date DEFAULT SYSDATE NOT NULL)

Explicaţie: În tabele **ANGAJAŢI** coloanele numele şi prenumele sunt definite cu constrângerea NOT NULL, deci aceste câmpuri trebuie obligatoriu completate.

În coloana data_ang se va introduce implicit valoarea returnată de funcţia SYSDATE data sistemului în cazul în care nu se completează acest câmp.

5. Constrângerile de verificare CHECK

Constrângerile de verificare a domeniului se folosesc pentru a exista siguranţa că introducerea într-o coloană (sau din mai multe coloane) satisface un set de criterii specificate. Constrângere CHECK se foloseşte pentru:

- verificarea valorilor minime sau .maxime (prevenirea apariţiei unei comenzi cu zero (0) articole, deşi zero este un număr permis sau notă unui elev nu poate fi mai mare de 10 sau numărul de absenţe nemotivate va fi cel mult egal cu numărul total de absenţe)
- specificarea intervalelor de timp (asigurarea că data unui transport este cel puţin egală sau ulterioară cu cea curentă, sau diferenţa dintre două date nu este mai mare de un an sau valoarea salariului trebuie să fie cuprinsă într-un domeniu de valori)
- acceptarea anumitor valori într-un anumit câmp (literele F şi M într-un câmp care precizează sexul sau numele şi prenumele unui ele trebuie să înceapă cu literă majusculă, restul literelor fiind mici sau câmpul bursier poate avea valorile ,'D' şi NULL)

Studiu de caz: Pentru a evita introducerea unei valori zero în câmpul „cantitate " aplicaţi o constrângere de verificare a valorii introduse în acest câmp.

CREATE TABLE **comenzi**

(nr_comandă mumber (3) PRIMARY KEY,

id_client number REFERANCE Clienţi(id_client),

id_produs number(4) UNIQUE

data date DEFAULT sysdate NOT NULL,

cantitate number(4) NOT NULL CHECK (cantitate>0),

preţ number(6,5) NOT NULL**);**

2.3.3. Modificarea structurii tabelelor

ALTER TABLE este instrucţiunea de modificare a structurii unei tabele

Operaţiile de modificare se execută cu ajutorul clauzelor prezentate în tabelul 2.30

- Adăugarea/ ştergerea unei coloane
- Modificarea tipului de data şi a dimensiunilor unei coloane
- Adăugarea/ ştergerea/ activarea/dezactivare unei constrângeri;

Tabel 2.30. clauzele comenzii ALTER TABLE

Acţiunea	Instrucţiunea generalizată
Adăugarea unei coloane la sfârşitul tabelului	**ADD**<nume_coloană>;
Ştergerea unei coloane	**DROP COLUMN** <nume_coloană>;
Modificarea unei coloane	**MODIFY** <nume_coloană><modificare>;
Adăugare unei constrângeri	**ADD CONSTRAINT <nume_constrângere>;**
Ştergerea unei constrângeri	**DROP CONSTRAINT <nume_constrângere>;**
Dezactivarea temporară şi reactivarea unei constrângeri	**DISABLI/ENABLE CONSTRAINT <nume_constrângere>[CASCADE]**
Ştergerea unei tabele	**DROP TABLE <nume_tabel>**

La modificarea coloanelor unei tabele trebuie respectate reguli:

- Poţi să măreşti lăţimea şi precizia unei coloane numerice;
- Poţi să măreşti dimensiunea unei coloane caracter dar, micşorarea dimensiunii o realizaţi doar dacă în coloana respectivă există numai valori de NULL);
- O coloană în care nu s-au introdus valori poate fi modificată (dimensiune şi tipul de dată)
- O coloană adăugată este iniţializată cu NULL dacă în tabelă sunt date

Tabel 2.31. Exemple instrucţiuni de modificare a coloanelor

Instrucţiunea	Exolicaţii
ALTER TABLE producători ADD telefon number(11);	La o tabelă care conţine date nu adăugăm o coloană cu restricţia NOT NULL !

ALTER TABLE producători ADD nr_com number(3) UNIQUE	Această instrucţiune adaugă la sfârşitul tabelului o coloană cu restricţia UNIQUE care poate avea oricâte valori de NULL
Instrucţiune	Explicaţie
ALTER TABLE producători ADD id_producător number(7) PRIMARY KEY;	Constrângerile NOT NULL sau PRIMARY KEY sunt permise în definirea unei coloane adăugate numai dacă tabela nu conţine date
ALTER TABLE elevi DROP COLUMN **bursieri**	instrucţiunea şterge coloană bursieri.
ALTER TABLE elevi DROP COLUMN **abs_nemotivate** CASCADE CONSTRAINS	pentru că există o constrângere pe această coloana instrucţiunea ar genera o eroare dacă nu se foloseşte clauza CASCADE CONSTRAINS
ALTER TABLE elevi MODIFY nume varchar2 (50)	Instrucţiunea modifică dimensiunea tipului coloanei de la varchar2 (20) la varchar2(50),
ALTER TABLE elevi MODIFY nume varchar2 (50)	Instrucţiunea modifică dimensiunea tipului coloanei de la varchar2 (20) la varchar2(50),
ALTER TABLE elevi MODIFY bursier DEFAULT 'F';	Instrucţiunea stabileşte o valoare implicită, dacă nu există deja una; această valoare nu va afecta liniile deja existente ci numai pe cele introduse în continuare

Tabel 2.32. Exemple instrucţiuni de modificare a unei constrângeri

Adăugarea unei constrângeri se realizează cu clauza ADD CONSTRAINT	
ALTER TABLE elevi ADD CONSTRAINT elevi_pk PRIMARY KEY (nr_matricol);	ALTER TABLE elevi ADD PRIMARY KEY (nr_matricol);
Ştergerea unei constrângeri se realizează cu clauza DROP CONSTRAINT	
ALTER TABLE elevi DROP PRIMARY KEY;	ALTER TABLE elevi DROP UNIQUE cnp;

Dezactivarea temporară şi apoi reactivarea unei constrângeri se realizează astfel	
ALTER TABLE <nume_tabelă> DISABLI/ENABLE CONSTRAINT <nume_constrângere>[CASCADE]	Clauza CASCADE precizează faptul că se activează/dezactivează constrângerile dependente Excepţie NOT NULL care se poate defini pentru o coloană numai prin modificarea coloanei

Ştergerea tabelelor (nu se aplică pentru un tabelul este relaţionat)

DROP TABLE <nume_tabel> instrucţiunea de ştergerea a tabelei

Această comandă şterge întregul tabel (nu numai conţinutul!)

Exemplu: DROP TABLE CLIENTI; instrucţiune şterge tabelul CLIENTI

Atenţie: Nu există posibilitatea recuperării datelor. Este bine să faceţi întotdeaina o copie a tabelului, folosind instrucţiune de creare a unui tabel cu subinterogări şi aplicaţi întotdeauna algoritmul de mai jos (tabel 2.32.), înainte de a şterge tabelul

Tabel 2.32 Operaţii pe care le foloseşti pentru a vizualiza structura unei tabele

	CE FACI?	**CUM FACI?**	**EXEMPLU**
1	**creezi o copie** a tabelului cu instrucţiunea	**CREATE TABLE** <copie_nume_tabel> **AS (SELECT * FORM** <nume tabel>**)**	**CREATE TABLE** copie_elevi **AS (SELECT * FORM** elevi
2	**verifici şi vizualizezi structura** copiei tabelului cu instrucţiunea	**DESCRIBE** <copie_nume_tabel>	**DESCRIBE** copie_elevi
3	**listezi datele** conţinute de copia tabelului	**SELECT *** **FROM** <copie_nume tabel>	**SELECT *** **FROM copie_elevi**

2.4. Limbaj de manipulare a datelor (DML)

Conţine instrucţiuni de adăugare şi ştergere a înregistrărilor şi a datelor dintr-un tabel

2.4.1.Instrucţiuni de adăugare/ştergere a înregistrărilor

Tabel 2.32. Comenzi de manipulare a datelor

Acţiunea	Comanda
Adăugarea unei înregistrări într-o tabelă	INSERT INTO <nume_tabela><lista coloane> VALUES <lista_valori>
Adăugarea mai multor înregistrări prin	INSERT INTO <nume_tabelă1><lista coloane>

preluare adatelor din alte tabele, cu ajutorul subinterogărilor	(SELECT <lista_coloane> FROM <<nume tabela2> WHERE <criteriu_selecţie>;
Ştergerea datelor dintr-o tabelă	DELETE FROM <nume_tabelă> WHERE <criteriu> Dcă este omisă clauza WHERE, se şterg toate liniile din tabela nominalizată
Ştergerea liniilor pe baza valorilor returnate de o subinterogare	DELETE FROM <nume_tabelă> WHERE <nume_coloană1>= (SELECT <nume_coloană1> FROM <nume_tabelă> WHERE <criteriu>)
Modificarea datelor dintr-o coloană	UPDATE <nume_tabelă> SET <coloana1>=valoare1, <coloana2>=valoare2, ……. WHERE <condiţie>

Tabel 2/33 Exemple de comenzi de manipulare a datelor

Instrucţiunea	**Acţiunea executată**
INSERT INTO producători (id_prod, nume_prod, localitate, telefon) VALUES <23, 'Romtelecom' , 'Sibiu', 0987658>	Inserarea unei linii în tabela producători
DELETE FROM produse WHERE id_producător=34	Stergerea liniilor care îndeplinesc criteriul id_producător-34

2.4.2 Vederi, secvenţe, indecşi, sinonime

Vederile (VIEW) sunt interogări care regăsesc în mod dinamic datele şi le prezintă în mod virtual. Vederile sunt tabele virtuale ca rezultat a unor interogări care regăsesc datele în mod dinamic atunci când sunt utilizate

Crearea unei vederi (View)

Pentru a crea o listă a clienţilor care au comandat cel puţin un produs

Vom scrie instrucţiunea

CREATE VIEW <nume_view>**AS** (subinterogare)

CREATE VIEW produsclienţi **AS (SELECT** numele, telefon, id_produs

FROM Clienţi, comenzi, articole

WHERE clienţi.id_client=comenzi.id_client)

AND comenzi.nr_comandă=comenzi.nr_comandă**);**

Utilizarea vederilor pentru:

- actualizarea datelor
- inserarea datelor
- ştergerea datelor
- modificarea datelor

SECVENŢELE sunt obiecte ale bazei de date care generează automat, în mod secvenţial , liste de numere care începe cu 1 şi are incrementul 1 (1,2,3,…)

CREATE SEQUENCE <nume _secvenţă>

START WITH 1 INCREMENT BY 1

INDEXURILE sunt folosite pentru sortarea logică a bazei de date pentru optimizarea operaţiilor de căutare şi sortare

CREATE INDEX <numele_index**>**

ON <nume_tabelă><lista_coloane>

SINONIME se defineşte de administratorul bazei de date, pentru un obiect al bazei de date în scopul simplificării referirii la acesta (pentru tabele, vederi, secvenţe, etc)

Exemplul: atribuirea unui sinonim tabelei cu numele tabea1 din chema unui utilizator user1 user1.tabela1.

CREATE SYNONYM <aprovizionare>

FOR user1.tabela1.

CREATE [PUBLIC] SYNONYM <aprovizionare> creează un sinonim pentru accesul

FOR user1.tabela1 tuturor utilizatorilor

2.5 Limbaj de control al datelor (DCL)

Securitatea bazei de date se asigură prin definirea şi restricţionarea drepturilor de acces pentru utilizatorii. Controlul securităţii Oracle se asigură prin specificarea :utilizatorilor bazei de date, schemelor, privilegiilor(drepturilor) şi rolurilor

Utilizatorii bazei de date şi schemele

Fiecare bază de date are o listă cu nume de utilizatori. Pentru a accesa baza de date un are un nume o parolă si un domeniu de securitate care determină: privilegiile, rolurile, spaţiul alocat pe disc, resursele pe care le utilizează.

Privilegiul este dreptul utilizatorului pentru a executa anumite instrucţiuni SQL:

- Privilegii de sistem permit administratorilor bazei de date modificarea datelor şi a structurii bazei de date.
- Privilegii de obiecte permit utilizatorilor să execute comenzi SQL numai în schema alocată.

Rolurile sunt grupe de privilegii care sunt acordate utilizatorilor din baza de date

Rolurile sunt grupuri de privilegii care sunt acordate utilizatorilor şi sunt create pentru a administra privilegiile pentru o aplicaţie de baze de date sau pentru administrarea privilegiilor pentru un grup de utilizatori. Se poate defini un rol pentru fiecare tip de utilizator al bazei de date.

Comenzile GRANT si REVOKE acordă şi revocă drepturi pentru un utilizator

GRANT CREATE SESSION TO<nume_utilizator>

REVOKE CREATE SESSION FROM <nume utilizator>

1. **Acordarea drepturilor de sistem pentru un utilizator**

GRANT CREATE SESSION , CREATE USER, CREATE TABLE TO<nume_utilizator>

2. **Acordarea drepturilor la nivel de obiect** permite utilizatorului să execute acţiuni asupra obiectelor bazei de date

GRANT SELECT, INSERT, UPDATE ON adm.elevi TO <nume utilizator> permite utilizatorului realizarea comenzilor interogare, inserare, modificare date

3. **Revocarea drepturilor**

REVOKE INSERT elevi FROM <nume_utilizator> revocă dreptul de inserare

4. **Crearea şi acordarea rolurilor**

CREATE ROL profi;

GRANT SELECT, INSERT,SELECT, UPDATE ON adm.elevi TO profi

GRANT profi TO <nume_utilizator1><nume utilizator2>

2.6 Limbajul de control al tranzacţiilor

O tranzacţie este formată din mai multe comenzi INSERT, UPDATE şi DELETE. Bazele de date relaţionale sunt astfel concepute încât datele sunt stocate în tabele diferite.

Tranzacţiile sunt blocuri (module) de instrucţiuni SQL care trebuie executate ca un întreg. Prelucrarea tranzacţiilor este necesară pentru conservarea integrităţii bazelor de date relaţionale, garantând faptul că grupurile de instrucţiuni SQL se execută în totalitate sau deloc.

Tranzacţiile sunt definite de instrucţiunile: **COMMIT, ROLLBACK, SAVEPOINT**

Utilizarea instrucţiunii COMMIT

Instrucţiunea COMMIT este folosită pentru a face permanentă o tranzacţie

Oracle impune marcarea explicită a începutului şi sfârşitului unei tranzacţii astfel:

```
BEGIN TRANSACTION
……
COMMIT  TRANSACTION
```

Toate instrucţiunile SQL cuprinse între BEGIN şi COMMIT TRANSACTION trebuie executate fie în totalitate, fie deloc

Exemplu: Considerăm următoarea tranzacţie:

```
BEGIN TRANSACTION
DELETE Articol WHERE ne_comandă=23
DELETE Comenzi WHERE nr_comandă=23
COMMIT  TRANSACTION
```

Explicaţie :În acest exemplu, comanda cu numărul 23 este ştearsă în totalitate din sistem. Deoarece aceasta implică actualizarea a două tabele ARTICOLE şi COMENZI se foloseşte un bloc de tranzacţie pentru a exista siguranţa că nu se realizează o ştergere parţială a comenzii numărul 23. Instrucţiunea COMMIT de la finalul tranzacţiei va scrie modificarea numai dacă nu se produce nici o eroare, dacă prima operaţie a fost executată, dar a doua a eşuat, operaţia de ştergere nu va fi scrisă

Utilizarea instrucţiunii ROLLBACK

Instrucţiunea ROLLBACK este folosită pentru finalizarea tranzacţiei şi anularea modificărilor efectuate în cadrul tranzacţiei

Instrucţiunea ROLLBACK <nume_punct_de_revenire> anulează modificările făcute până la punctul de revenire definit în tranzacţie

Exemplu: DELETE from Comenzi ROLLBACK

Explicaţie: în acest exemplu o operaţie de ştergere din tabele COMENZI este mai întâi efectuată şi apoi anulată prin intermediul comenzii ROLLBACK

Utilizarea comenzii SAVEPOINT

În tranzacţie se pot definii puncte de revenire, de întoarcere, de salvare până la care se execută o revenire prin intermediul unei comenzi ROLLBACK.

Instrucţiunea SAVEPOINT <nume_punct_de_revenire> defineşte un punct de revenire. Este utila această definire a punctelor de revenire pentru tranzacţiile mari. În cazul în care apare o greşeală nu se renunţă la întreaga tranzacţie ci se revine la un punct de salvare

Exemplu: SAVEPOINT stergere1;

Fiecare punct de salvare primeşte un nume unic, astfel încât, atunci când se execută o revenire (anularea modificărilor făcute) Oracle ştie unde este punctul în care se opreşte revenirea. Pentru revenirea la punctul ”stergere1” se scrie instrucţiunea ROLLBACK TRANSACTION stergere1

În continuare prezint un exemplu complet de tranzacţie SQL_server

BEGIN TRANSACTION

INSERT INTO Clienti (id_client, numele);

VALUES(25,’Marinescu’);

SAVE TRANSACTION începe_comanda; (*este introdus un punct de salvare)

INSERT INTO Comenzi (nr_comanda, id_client, data)

VALUES(56, 25, 10/06/2010)

If @@ERROR<>0 ROLLBACK începe_comanda (*dacă introducerea datelor eşuează se revine la punctul de salvare cu numele începe_comanda)

INSERT INTO Articole (nr_comanda, nr_articol, id_produs, Cantitate, Preţ)

VALUES(56, 22, 11, 6, 34)If @@ERROR<>0 ROLLBACK începe_comanda COMMIT TRANSACTION Explicaţie:Un set de patru instrucţiuni INSERT sunt incluse într-o tranzacţie După prima instrucţiune este definit un punct de salvare, astfel încât, dacă una dintre

următoarele instrucţiuni INSERT eşuează, tranzacţia va fi anulată numai până la acel punct. Dacă variabila definită @@ERROR returnează o altă valoare decât zero înseamnă că s-a produs o eroare şi tranzacţia este anulată până la punctul de salvare denumit ”începe_comanda”. Dacă întreaga tranzacţie este prelucrată instrucţiunea COMMIT salvează dalele în tabele.