

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

CUPRINS

CAPITOLUL 1 SISTEME DE GESTIUNE A BAZELOR DE DATE	1-5
1.1 Arhitectura sistemelor de gestiune a bazelor de date	1-5
1.2 Obiectivele și funcțiile unui sistem de gestiune a bazelor de date	1-6
1.3 Evoluția și clasificarea sistemelor de gestiune a bazelor de date	1-8
1.4 Conceptul de bază de date relațională	1-11
1.5 Terminologie specifică bazelor de date relaționale	1-14
1.6 Sisteme de gestiune a bazelor de date relaționale în Oracle	1-16
1.7 Sisteme de gestiune a bazelor de date relaționale în FoxPro	1-18
CAPITOLUL 2 CONCEPTE ȘI ELEMENTE DE BAZĂ ÎN FOXPRO.....	2-22
2.1 Descrierea mediului integrat	2-25
2.2 Tipuri de date în FoxPro, operatori, funcții	2-26
2.3 Funcții pentru conversii între tipuri de date	2-34
2.4 Zone de lucru	2-35
2.5 Construirea bazelor de date relaționale. Componente	2-36
2.6 Comenzi pentru vizualizarea și modificarea datelor din tabele	2-40
2.7 Exerciții	2-44
CAPITOLUL 3 INDEXAREA ȘI RELAȚIONAREA TABELELOR, INTEGRITATEA REFERENȚIALĂ	3-46
3.1 Indexarea tabelelor	3-46
3.2 Relaționarea tabelelor	3-50
3.3 Integritatea referențială	3-53
3.4 Funcții. Variabile de memorie. Macrosubstituția	3-54
3.5 Crearea și modificarea programelor	3-57
3.6 Programare structurată. Proceduri	3-58
3.7 Comenzi de intrare/iesire	3-60
3.8 Controlul fluxului	3-65
3.9 Comenzi SQL	3-67
3.10 Depanarea programelor	3-69
3.11 Instrumente WIZARD	3-71
3.12 Exerciții	3-74
CAPITOLUL 4 PROGRAMAREA ORIENTATA PE OBIECT-CLASE ȘI OBIECTE ÎN VISUAL FOX PRO.....	4-78
4.1 Crearea claselor	4-81
4.2 Crearea formularelor	4-85
4.3 Utilizarea controalelor predefinite	4-87
4.4 Generatorul de rapoarte	4-99
4.5 Constructorul de meniuri	4-105
4.6 Constructorul de proiecte	4-108
4.7 Exerciții:	4-110

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

CAPITOLUL 5 LIMBAJUL SQL IMPLEMENTAT ÎN ORACLE.....	5-111
5.1 Soluția completă Oracle	5-111
5.2 Comenzi SQL*Plus pentru fișiere	5-114
5.3 Instrucțiuni SQL	5-115
5.4 Sintaxa de bază a instrucțiunilor SQL	5-116
5.5 Crearea și gestionarea tabelelor	5-118
5.6 Comanda CREATE TABLE	5-120
5.7 Tabele din baza de date Oracle	5-121
5.8 Interogarea dicționarului de date	5-122
5.9 Tipuri de date	5-123
5.10 Comanda ALTER TABLE	5-127
5.11 Ștergerea unei tabele	5-131
5.12 Modificarea numelui unui obiect	5-132
5.13 Trunchierea unei tabele	5-132
5.14 Includerea constrângerilor	5-133
5.15 Constrângerea NOT NULL	5-136
5.16 Constrângerea UNIQUE KEY	5-137
5.17 Constrângerea PRIMARY KEY	5-139
5.18 Constrângerea FOREIGN KEY	5-139
5.19 Constrângerea CHECK	5-141
5.20 Adăugarea unei constrângerii	5-142
5.21 Ștergerea unei constrângerii	5-143
5.22 Dezactivarea constrângerilor	5-143
5.23 Activarea constrângerilor	5-144
5.24 Exerciții	5-147
CAPITOLUL 6 EXPRESII ARITMETICE. OPERATORI.	
RESTRICTIONAREA ȘI SORTAREA DATELOR.....	6-149
6.1 Expresii aritmetice	6-149
6.2 Definirea alias-urilor pentru coloane	6-153
6.3 Operatorul de concateneare	6-154
6.4 Afisarea structurii unei tabele	6-157
6.5 Restricționarea și sortarea datelor	6-158
6.6 Clauza WHERE	6-159
6.7 Operatori de comparație	6-161
6.8 Operatori logici	6-165
6.9 Clauza ORDER BY	6-169
6.10 Exerciții	6-172
CAPITOLUL 7 FUNCȚII DE UN SINGUR RÂND	7-174
7.1 Funcții de un singur rând	7-175
7.2 Funcții pentru caractere	7-177
7.3 Funcții pentru valori numerice	7-181
7.4 Utilizarea datelor calendaristice	7-183

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

7.5	Funcții pentru date calendaristice.....	7-184
7.6	Funcții pentru conversia tipului de date.....	7-187
7.7	Funcții diverse	7-196
7.8	Imbricarea funcțiilor	7-203
7.9	Exerciții	7-204
CAPITOLUL 8 AFIȘAREA DATELOR DIN TABELE MULTIPLE.....		8-205
8.1	Definirea JOIN-urilor.....	8-206
8.2	Produsul Cartezian	8-206
8.3	Echi-join	8-208
8.4	Non-echi-join	8-212
8.5	Outer-join	8-214
8.6	Self – Join.....	8-216
8.7	Definirea join-urilor folosind sintaxa SQL 1999	8-218
8.8	Exerciții	8-226
CAPITOLUL 9 FOLOSIREA FUNCȚIILOR DE GRUP		9-231
9.1	Ce sunt funcțiile de GRUP ?	9-231
9.2	Folosirea funcțiilor AVG, SUM, MIN, MAX	9-232
9.3	Folosirea funcției COUNT.....	9-233
9.4	Funcțiile de grup și valorile Null.....	9-234
9.5	Crearea grupurilor de date	9-235
9.6	Gruparea datelor după mai multe coloane	9-238
9.7	Interogări ilegale în folosirea funcțiilor de grup.....	9-240
9.8	Excluderea rezultatelor obținute folosind clauza Group.....	9-242
9.9	Imbricarea funcțiilor de grup	9-244
9.10	GROUP BY cu operatorii ROLLUP și CUBE.....	9-244
9.11	Operatorul ROLLUP	9-245
9.12	Operatorul CUBE.....	9-247
9.13	Funcția GROUPING	9-249
9.14	GROUPING SETS.....	9-250
9.15	Composite Columns	9-254
9.16	Concatenated Groupings.....	9-257
9.17	Exerciții	9-259
CAPITOLUL 10 SUBINTEROGARI.....		10-265
10.1	Folosirea unei subinterrogări pentru a rezolva o problema	10-265
10.2	Tipuri de subinterrogări.....	10-267
10.3	Subinterrogari single-row	10-268
10.4	Utilizarea funcțiilor de grup într-o subinterrogare	10-269
10.5	Erori ce pot apărea la folosirea subinterrogărilor.....	10-270
10.6	Subinterrogări multiple-row.....	10-272
10.7	Utilizarea operatorului ANY în subinterrogările multiple-row	10-273
10.8	Utilizarea operatorului ALL în subinterrogările multiple-row	10-273
10.9	Returnarea valorilor nule în rezultatul subinterrogării.....	10-274

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

10.10	Subinterrogari de coloane multiple	10-275
10.11	Compararea coloanelor (pereche și nepereche).....	10-275
10.12	Folosirea unei subinterrogări în clauza FROM.....	10-276
10.13	Expresii scalare returnate de subinterrogări.....	10-277
10.14	Subinterrogări corelate	10-279
10.15	Folosirea operatorului EXISTS	10-282
10.16	Folosirea operatorului NOT EXISTS	10-284
10.17	Clauza WITH	10-284
10.18	Interrogări ierarhice.....	10-286
10.19	Parcurserea arborelui – punctul de start	10-288
10.20	Exerciții	10-295
CAPITOLUL 11 INSTRUCȚIUNI PENTRU MANIPULAREA DATELOR....		11-302
11.1	Introducerea datelor-comanda INSERT	11-303
11.2	Modificarea datelor - comanda UPDATE	11-308
11.3	Ștergerea datelor - comanda DELETE.....	11-311
11.4	Instrucțiunea MERGE	11-314
11.5	Tranzacții	11-316
11.6	Consistența la citire	11-321
11.7	Correlated UPDATE	11-323
11.8	Correlated DELETE	11-324
11.9	Exerciții	11-333
CAPITOLUL 12 ANEXA 1.....		12-337
STRUCTURA TABELELOR FOLOSITE ÎN CARTE ȘI DATELE STANDARD CONTINUTE DE ACESTEA		12-337
BIBLIOGRAFIE		12-343

Capitolul 1 Sisteme de gestiune a bazelor de date

Sistemele de Gestiune a Bazelor de Date (SGBD) reprezintă componenta software a unui sistem de baze de date care asigură independentă, relațiile logice între date și o redundanță minimă a acestora. Ele trebuie să permită dezvoltarea rapidă și la un cost avantajos a programelor de aplicații pentru exploatarea datelor dintr-o structură complexă, precum și accesul rapid la date și asigurarea securității lor. Altfel spus, SGBD-ul este un ansamblu de programe care permite utilizatorilor să interacționeze cu o bază de date, concepută de regulă pentru volume mari de date, a căror gestiune impune nu numai o riguroasă structurare dar și o accesare și prelucrare rațională.

1.1 Arhitectura sistemelor de gestiune a bazelor de date

Datorită dezvoltării tot mai accentuate a IT-ului (Information Technology) în majoritatea domeniilor de activitate și datorită extensiei sferei problemelor rezolvate cu ajutorul tehnicii de calcul, a apărut ca o necesitate specializarea pachetelor de programe în funcție de domeniile abordate: matematică, tehnică, economie, proiectare, comunicație etc.

Sistemele de Gestiune a Bazelor de Date sunt sisteme informaticе software specializate în stocarea și prelucrarea unui volum mare de date. Sunt implicate două concepte: "baza de date" și "gestiune". Prin "baza de date" se înțelege: datele de prelucrat și modul de organizare a acestora pe suportul fizic de memorare și prin "gestiune" totalitatea operațiilor ce se vor aplica asupra datelor. Dintre avantajele organizării informațiilor în baza de date față de fișierele clasice (de tip ASCII, binare, etc.) putem aminti redundanța minimă a informațiilor, accesul mai ușor la date și posibilitatea abordării domeniului implementat din punct de vedere sistemic (ca un sistem unitar).

Piața SGBD-urilor este dominată încă de SGBD-urile relaționale și distribuite, deși se observă o dezvoltare a SGBD-urilor orientate obiect. Lucrările practice de baze de date din această carte vor cuprinde prezentarea SGBD-ului FoxPro și Oracle.

Teoria și practica SGBD-urilor oferă diferite arhitecturi diferențiate în funcție de componente, limbajele utilizate și posibilitățile de prelucrare a datelor, existând totuși preocupări de standardizare a acestora.

În general, în arhitectura unui SGBD intră cel puțin 5 clase de module:

- *Programele de gestiune a bazelor de date.* Această clasă de module realizează accesul fizic la date ca urmare a unei comenzi primite printr-un program de aplicații sau interactiv de la tastatură;
- *Limbajul de definire a datelor (LDD).* Este componenta care permite

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

traducerea (compilarea sau interpretarea, după caz) și descrierea naturii datelor și a legăturilor logice dintre ele, fie la nivel global (sub forma *schemei conceptuale*), fie la nivelul specific fiecărei aplicații (sub forma *schemei externe sau sub-schemei*). Aceste definiții se memorează într-un fișier special numit *Dicționarul de date*.

- *Limbajul de manipulare a datelor (LMD)*. Această componentă permite gestionarea și actualizarea datelor dintr-o bază (scrierea unor noi date, modificarea valorii unor date, ștergerea unor date perimate sau eronate). Comenzile acestui limbaj depind de SGBD-ul utilizat (cu limbaj gazdă sau SGBD autonome). În SGBD-urile cu limbaj gazdă enunțurile trebuie incluse într-un program scris cu un limbaj gazdă: PL/1, ASSEMBLER, C etc. În SGBD-urile autonome se dispune de un limbaj la care se atașează un limbaj de interogare.
- *Utilitarele de întreținere a bazei de date*. Un SGBD trebuie să ofere o gamă variată de programe utilitare care să permită gestionarea de către operator a bazei de date. Utilitarele variază de la un sistem la altul și depind de complexitatea SGBD-ului. Acestea pot efectua următoarele operații: crearea versiunii inițiale a bazei de date și încărcarea acesteia folosindu-se fie o copie creată anterior, fie date neorganizate; crearea și actualizarea jurnalelor tranzacțiilor realizate asupra bazelor de date: reorganizarea bazei de date pentru recuperarea spațiului nefolosit; reorganizarea structurii fizice și logice după fiecare tranzacție; restructurarea bazei de date după un incident logic sau fizic, cu refacerea stării anterioare; diverse statistici ce permit cunoașterea activității și utilizării bazei de date; actualizarea schemei și sub-schemei fără rescrierea și compilarea lor; detectarea “spărgătorilor” regulilor de integritate definite, fără a fi necesară intrarea în baza de date; realizarea unei copii permanente a bazei de date în scopuri de securitate.
- *Componentele de control ale programelor de aplicații*. Acestea constituie mijloace de prevenire și corectare a anumitor erori ce pot să apară în condiții de exploatare “multi-utilizator”.

1.2 Obiectivele și funcțiile unui sistem de gestiune a bazelor de date

Succesul unui SGBD este susținut prin realizarea următoarelor obiective:

- **asigurarea independentei datelor**. Independența datelor poate fi definită drept “imunitatea” programelor de aplicații la schimbarea structurii de memorare și /sau a strategiei de acces. Independența

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

datelor trebuie urmărită atât la nivel fizic cât și la nivel logic. *Independența fizică* asigură modificarea datelor și a tehnicilor fizice de memorare, fără rescrierea programelor de aplicații. *Independența logică* oferă posibilitatea adăugării de noi articole sau extinderea structurii globale fără a necesita rescrierea programelor;

- **asigurarea integrității datelor** prin existența unor proceduri de validare sau a unor protocoale de control concurrent, precum și a unor proceduri de refacere a bazei de date după incidente;
- **asigurarea unei redundanțe minime și controlate a datelor** prin definirea unui element de structură, cu o cantitate cât mai mică de date, evitându-se în același timp ambiguitatea;
- **asigurarea unor facilități sporite de utilizare a datelor** prin: folosirea datelor de către mai mulți utilizatori în diverse aplicații, accesul simplu și multicriterial al utilizatorilor la date, fără a fi necesară cunoașterea structurii întregii baze de date; existența unor limbaje performante de interogare etc.;
- **asigurarea partajării datelor**, adică asigurarea accesului mai multor utilizatori la aceleși date pe baza unor criterii de prioritate și dezvoltarea unor aplicații fără a se modifica structura bazei de date;
- **asigurarea securității datelor** prin intermediul unor canale corespunzătoare și definirea unor restricții de autorizare la accesarea datelor.

Plecând de la aceste obiective, rezultă că orice SGBD trebuie să îndeplinească următoarele funcții: **de descriere, de manipulare, de utilizare**.

Funcția de descriere permite definirea structurii bazei cu ajutorul limbajului special de descriere a datelor, stabilind criterii de validare a acestora, metode de acces și de asigurare a confidențialității și integrității lor. Toate aceste elemente se regăsesc în ceea ce se numește *schema bazei de date*. Definițiile se stochează într-un ansamblu de tabele, memorate în dicționarul de date.

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

Funcția de manipulare asigură prin intermediul limbajului special derularea următoarelor activități: încărcarea bazei de date, adăugarea de noi înregistrări, stergerea unor înregistrări, editarea totală sau parțială a unor înregistrări, ordonarea înregistrărilor, căutarea logică sau fizică a înregistrărilor etc.

Funcția de utilizare permite comunicarea între utilizatori și baza de date prin intermediul unor interfețe avantajoase utilizatorilor. În felul acesta se creează un mediu favorabil utilizatorului care la ora actuală beneficiază de prelucrarea în timp real, de arhitecturile client-server, servicii Internet etc. În cadrul realizării acestei funcții interacționează diverși utilizatori, literatura de specialitate oferind mai multe clasificări sau grupări. Dintre acestea prezentăm în continuare doar câteva astfel de grupări.

Raportul ANSI/SPARC 1975 prezintă trei categorii de utilizatori (roluri umane) ce definesc schemele dintr-o arhitectură de sistem bazat pe SGBD:

- *persoana sau grupul de persoane care definește schema conceptuală a bazei de date.* Această schemă furnizează o viziune pe termen lung și este baza pentru declarațiile de securitate-integritate și standardizare impuse celoralte tipuri de utilizatori;
- *administratorul bazei de date* care are responsabilitatea definirii schemei interne a bazei de date și a întreținerii acesteia. În același raport sunt prezentate trei categorii de administratori: administratorul structurii organizaționale care asigură gestionarea globală a aplicațiilor curente și identificarea celor viitoare; administratorul aplicațiilor care are rolul de a dezvolta schemele externe (sub-schemele) pentru aplicațiile utilizator; administratorul de date care operează la nivelul schemei de date precizând necesarul și disponibilitatea datelor;
- *programatorii de aplicații și utilizatorii finali* care comunică cu SGBD-ul prin intermediul limbajului de manipulare sau a limbajului de interogare.

1.3 Evoluția și clasificarea sistemelor de gestiune a bazelor de date

Perfecționarea SGBD-urilor a avut loc în paralel cu evoluția echipamentelor de culegere, memorare, transmitere și prelucrare a datelor, pe de o parte, și cu evoluția metodelor și tehnicilor de organizare a datelor, pe de altă parte.

Literatura și practica de specialitate oferă o mare diversitate de sisteme

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

de gestiune a bazelor de date, care sintetic pot fi prezentate în trei generații.

Prima generație este situată în timp la sfârșitul anilor '60 și începutul anilor '70 și include modelul ierarhic și modelul rețea de organizare a datelor în baze de date.

Modelul ierarhic are o structură de tip arborescent, legată puternic de principiile de descriere a datelor specifice limbajelor din generația corespunzătoare. Fiecare nivel de date este constituit din una sau mai multe grupe de date care se pot și ele descompune la rândul lor.

Modelul rețea încearcă să înlăture lipsurile modelului precedent, propunând o structură de date mai bogată pe baza legăturilor posibile și conducând la construirea unei rețele între acestea.

Generația a doua, situată în timp la sfârșitul anilor '80 și începutul anilor '90 cuprinde în principal modelele relaționale. Structura datelor este formată dintr-un ansamblu de relații sau tabele fără legături fizice între ele. Apropierea dintre date este obținută prin utilizarea unei "algebrelor" între tabele, considerate ca operanți. Din această generație fac parte: IBM DB2, Oracle, Microsoft SQL Server, PARADOX, FoxPro etc.

Generația a treia apărută la începutul anilor '90. Aceste sisteme se bazează pe principii mult mai complexe decât precedentele și permit gestionarea unor informații foarte variate. Se pot încadra în această generație 4 subclase de SGBD: orientate obiect, funcționale, deductive și multimedia.

Sistemele orientate obiect permit descrierea elementelor unei baze utilizând concepții abordării obiectuale care ține seama de aspectele statice și dinamice ale obiectelor. Această abordare se caracterizează, în principal, pe operația de *incapsulare*, adică pe reunirea în aceeași unitate a caracteristicilor statice și dinamice ale obiectelor de gestionat. În plus, este posibilă reutilizarea obiectelor deja definite. Această caracteristică reduce considerabil programarea "defensivă" datorită descrierii intrinseci făcută obiectelor. Obiectele definite în bază au capacitatea de a transmite descendenților, ansamblul caracteristicilor lor prin "moștenire".

Sistemele funcționale au la bază noțiunile de entitate și funcție și au fost introduse în 1979 de Shipman. Pentru fiecare entitate (obiect) de un anumit tip există o colecție de funcții care sunt aplicate aceluui obiect, definind atributele obiectului și exprimând asociările dintre entități. Modelul funcțional devine operațional prin intermediul SGBD-urilor funcționale care oferă mecanisme și instrumente de descriere și manipulare a datelor prin intermediul limbajelor funcționale.

Sistemele deductive utilizează reguli de inferență pentru a exprima situații bine sau mai puțin bine definite din mediul unei organizații. Ele permit

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

să se răspundă cererilor care comportă exprimarea într-o logică mai mult sau mai puțin vagă. În general astfel de baze de date sunt cuplate cu sistemele expert sau cu un nivel din baza regulilor scrise în PROLOG, LISP, sau într-un alt limbaj de inteligență artificială.

Sistemele multimedia permit stocarea și administrarea altor informații decât cele clasice precum: imagini, fotografii, muzică, sunete etc. Sistemele din această categorie, care se comercializează deja, permit organizarea de "birouri fără hârtie", propunând gestiunea diferitelor documente (imprimeate) pe discul optic cu proceduri de înregistrare prin scanare precum și prin proceduri de consultare și manipulare.

Preocupările proiectanților de SGBD sunt concentrate spre construirea bazelor de date pe modele obiectuale, deductive și multimedia, fiind propuse mai multe soluții de combinare și extindere a modelelor prezentate: înglobarea în SGBD-urile relaționale a elementelor ce definesc tehnologia actuală, persistența obiectelor prin extinderea actualelor sisteme obiectuale, integrarea tehnologiei semantice și obiectuale etc.

Clasificarea SGBD-urilor

Abordarea sistemică a problematicii referitoare la SGBD-uri presupune și clasificarea acestora, mai cu seamă că literatura de specialitate prezintă un număr deosebit de mare de astfel de software-uri. Există mai multe clase de SGBD private din diverse puncte de vedere: modelul de organizare folosit (ierarhice, rețea, relaționale, orientate obiect etc.), distribuirea resurselor (integrate și distribuite teritorial), destinație (publică și privată) și tehnica de prelucrare (pe loturi, interactivă, mixtă). Ne oprim, în continuare, doar asupra câtorva astfel de criterii de clasificare.

După sistemul de calcul pe care se implementează SGBD-urile pot fi:

- pentru *mainframe-uri*;
- pentru *minicalculatoare*;
- pentru *microcalculatoare*.

La ora actuală tendința este de generalizare și standardizare a SGBD-urilor conferindu-lui-se atributul de "cross platforma" pentru a putea fi rulate pe cât mai multe medii de operare (DOS, WINDOWS, MAC, UNIX, Linux, OS/2 etc.).

În funcție de limbajul utilizat există două clase de SGBD-uri:

- *cu limbaj gazdă*: asigură crearea, actualizarea și interogarea bazei de date utilizând limbiile de nivel înalt propriu sistemului de calcul pe

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

care se implementează baza de date. Prezintă avantajul exploatării facilităților limbajului de nivel înalt și dezavantajul complexității formulării cererilor de prelucrare;

- *cu limbaj autonom (propriu)*: folosesc limbaje speciale care sunt independente de limbajul gazdă și chiar față de sistemul de calcul ceea ce asigură un grad ridicat de portabilitate. Avantajul acestor SGBD-uri concretizat în larga lor utilizare, îl reprezintă simplitatea formulării cerințelor.

În funcție de organizarea și structurarea datelor gestionate SGBD-urile sunt:

- *ierarhice și rețea*;
- *relaționale*;
- *obiectuale*;
- *obiectual-relaționale*.

Din punct de vedere a modului de localizare a bazelor de date SGBD-urile sunt:

- *centralizate*, care presupun ca datele să fie concentrate într-un singur loc pe un server, ca sursă unică de informare și/sau raportare (VisualFoxPro, Access, etc.);
- *distribuite*, care presupun ca datele să fie repartizate geografic în mai multe puncte în funcție de locul lor de producere, cu posibilitatea accesării datelor din orice punct definit (Oracle, IBM DB2, Informix).

La ora actuală datorită proliferării prelucrărilor în rețea de calculatoare tot mai multe SGBD-uri dispun de o componentă de gestiune distribuită a datelor.

1.4 Conceptul de bază de date relațională

Dr. E.F. Codd a propus modelul relațional pentru baze de date în 1970 (http://www.wikipedia.org/wiki/Edgar_F._Codd), model ce reprezintă baza pentru sistemele de gestiune a bazelor de date (DBMS).

Principiile modelului relațional au fost subliniate pentru prima dată de Dr. E.F. Codd în lucrarea numită “Un model relațional de date pentru bănci de date mari”. Cele mai populare modele ale vremii erau cele ierarhice și de rețea sau chiar fișierele simple de date. Apoi DBMS-urile au devenit foarte populare pentru ușurința în utilizare și flexibilitatea lor.

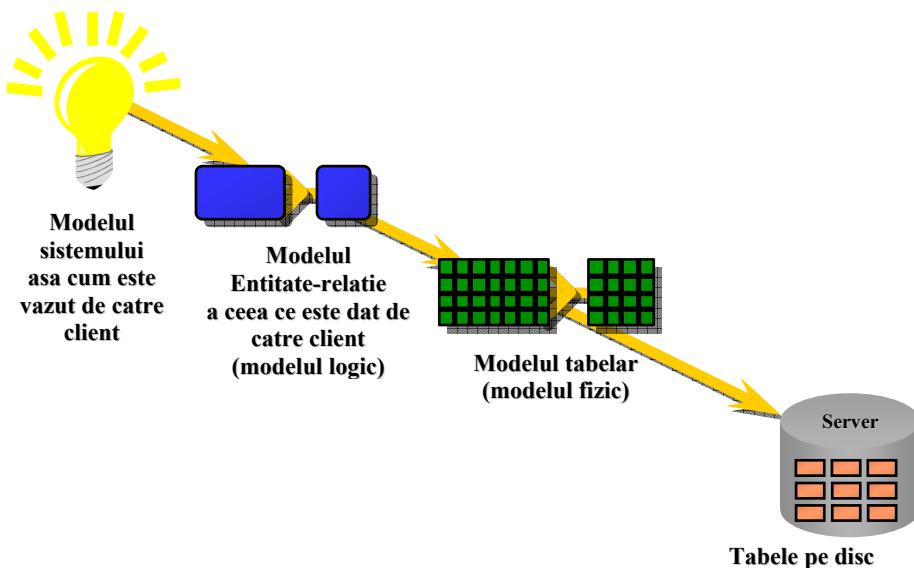
Componentele modelului relațional sunt:

- Colecție de obiecte și relații care stochează date;
- Un set de operatori care acționează asupra relațiilor pentru a produce alte relații;
- Reguli de integritate a datelor pentru consistență și acuratețe.

Definirea unei baze de date relaționale

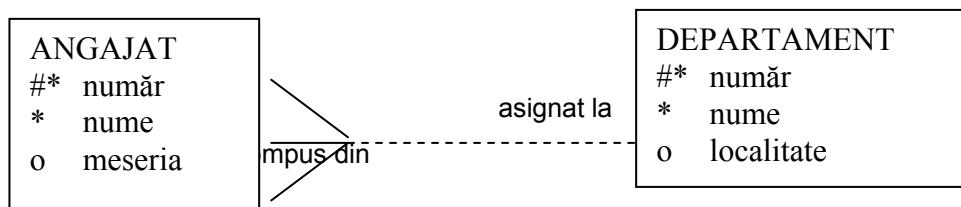
O baza de date relațională folosește relații sau tabele bi-dimensionale pentru stocarea informațiilor. De exemplu, pentru stocarea informațiilor despre angajații unei companii, într-o bază de date relațională veți găsi informația stocată în: tabela angajați, tabela departamente și tabela salarii.

Modelele de date sunt dezvoltate de proiectanți pentru a explora idei și pentru a îmbunătăți înțelegerea proiectării bazei de date. Modelele ajută în comunicarea conceptelor în mintile oamenilor. Ele pot: comunica, pune pe categorii, descrie, specifica, investiga, analiza, imita și implica. Scopul este de a produce un model care să se potrivească și să răspundă tuturor cerințelor și care să fie înțesă de utilizatorul final, care să conțină detalii suficiente pentru ca un programator să construiască sistemul de baze de date și/sau să dezvolte o aplicație care să lucreze cu baze de date.



Modelul Entitate Relație – permite crearea unei diagrame entitate relație pe baza specificațiilor informaționale ale problemei.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



Scenariu:

“ ... se asigneză unul sau mai mulți angajați la un departament ...”

“... câteva departamente nu au nici un angajat ...”

Într-un sistem real, data este divizată în categorii discrete sau entități. Un model entitate relație (ER) este o ilustrare a entităților dintr-o problemă și a relațiilor dintre ele. Un model ER este derivat din specificațiile informaționale ale problemei și este construit în faza de analiza a sistemului. Modelul ER separă informațiile cerute de problema de activități. Chiar dacă activitățile problemei se schimbă, tipul informațiilor trebuie să rămână constant. De aceea structurile de date trebuie să rămână constante.

Avantajele modelului ER:

- Documentează necesarul de informații pentru organizație într-o formă clară și precisă;
- Prezintă o imagine clară a scopului cerințelor informaționale și ușor de înțeles pentru proiectarea bazei de date;
- Oferă un mediu efectiv de integrare a aplicațiilor multiple.

Componente cheie:

Entitatea: Un obiect sau un grup de obiecte de același tip (cu aceleași proprietăți), cu o anumită semnificație, despre care este necesar să fie cunoscute informații. Exemplu: departamente, angajați, comenzi.

Atribut: Un aspect care descrie sau califică o entitate. Exemplu: pentru entitatea Angajat atributul sunt: marca, meseria, data de angajare, departamentul. Fiecare atribut poate fi obligatoriu sau optional.

Relație: Un nume de asociere dintre entități care arată optionalitatea sau gradul asocierii. Exemplu: angajați și departamente, comenzi și articole.

Identifier unic (UID – Unique IDentifier) este orice combinație de atribut sau/și relații care servește la distingerea aparițiilor unei entități. Fiecare apariție a entității trebuie să fie identificată unic.

1.5 Terminologie specifică bazelor de date relationale

O baza de date relatională conține una sau mai multe tabele. O tabelă reprezintă structura de bază a unui SGBDR (Sistem de Gestire a Bazelor de Date Relaționale). O tabelă conține toate datele necesare despre un aspect al lumii reale. Se consideră exemplul tabelei Angajați.

Noțiuni:

- Un singur *rând* sau o *tuplă* reprezintă informația necesară pentru un angajat specificat. Fiecare rând din tabelă trebuie identificat de o *cheie primară*, care nu permite rânduri duplicate. Ordinea rândurilor nu este semnificativă.
- O *coloană* sau un *atribut* conține marca angajatului care este și *cheie primară*. Marca (cheia primară) identifică în mod unic un angajat în tabela EMP. O cheie primară trebuie să conțină o valoare. Alte tipuri de coloane reprezintă alte informații (de ex: funcția). Ordinea coloanelor nu este importantă. Coloana care conține numărul de departament este numită și *cheie externă*. Cheia externă conține coloane care definesc modul în care sunt în relație tabele diferite. O *cheie externă* referă o cheie primară sau o cheie unică din alt tabel.
- O coloană reprezintă un *domeniu de date* dintr-o tabelă. În exemplul dat „meseria” este coloana ce conține toate meseriile pentru toți angajații.
- Un *câmp* poate fi găsit la intersecția dintre un *rând* și o *coloană*. Poate conține o singură valoare.
- Câmpurile pot să nu conțină valori. Acestea se numesc valori *null*.

Referirea mai multor tabele - Fiecare tabelă conține date care descriu exact o entitate. De exemplu tabela Angajați conține informații despre angajați. Deoarece datele despre entități diferite sunt stocate în tabele diferite, este necesară combinarea a două sau mai multe tabele pentru a afla răspunsul la o interogare. De exemplu dacă vrei să aflați localitatea în care este situat departamentul în care lucrează un anumit angajat sunt necesare tabelele Angajați și Departamente. Un SGBDR permite combinarea tabelelor diferite folosind chei externe. O *cheie externă* este o coloană sau o combinație de coloane care referă o cheie primară din același tabel sau din unul diferit.

Proprietățile bazelor de date relationale

O baza de date relatională :

- Poate fi accesată și modificată prin execuția instrucțiunilor specifice;

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

- Conține o colecție de tabele fără pointeri fizici;
- Folosește un set de operatori.

Într-o bază de date relațională nu trebuie specificată calea de acces la tabele și nu trebuie cunoscut modul de aranjare fizic.

Obiecte - Un obiect este considerat ca o reprezentare a unui lucru din lumea reală. Câteva definiții:

- “Un obiect este un pachet software care conține o colecție de proceduri (metode) înrudite și date (variabile)” - David Taylor, “Object-Oriented Technology: A Manager’s Guide” Addison-Wesley 1981
- “Un obiect este un concept, abstracție sau lucru cu margini și înțelesuri fragile pentru problema de rezolvat” - James Rumbaugh, “Object-Oriented Modeling and Design” Prentice-Hall 1991

Modele obiect:

- obiectele modelează o problemă spre rezolvare;
- modelul este definit în termeni de interacțiune între obiecte;
- modelele obiect se aseamănă cu lumea reală;
- când se lucrează cu modele ne axăm mai mult în termenii aplicației și mai puțin la arhitectura sistemului de operare și cerințele mediului de dezvoltare.

Caracteristicile sistemelor obiectuale:

- Prezintă informația în forma obiect;
- Clasifică obiectele în tipuri de obiecte;
- Moștenește atribute și cod;
- Ascunde date, cod și atribute;
- Interacționează cu alte obiecte;
- Recunoaște diferite obiecte fără analiză;
- Interpretează aceeași comandă în moduri diferite;

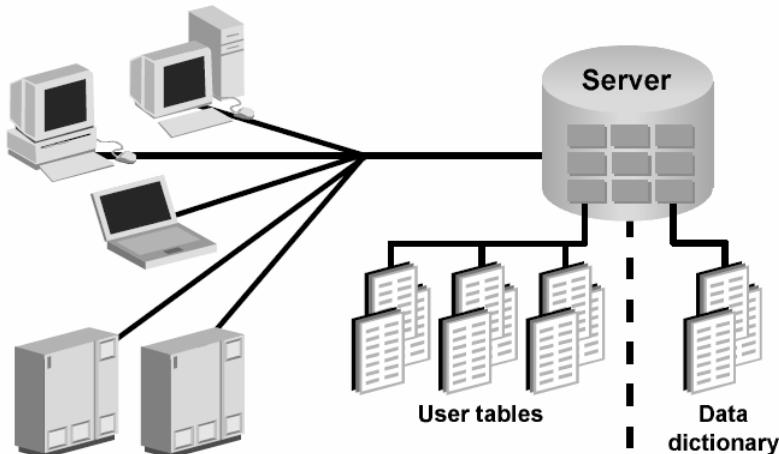
Alte caracteristici:

- Reprezintă informația ca lucruri conținute întrinsec;
- Clasifică obiectele în tipuri de obiecte organizate în ierarhii arborescente, unde un tip de obiect poate fi alt fel decât alt tip de obiect (object type metadata);
- Obiectele pot moșteni trăsăturile tipului sau de obiecte (inheritance);
- Pot ascunde date și procese referitoare la obiect în cadrul obiectului (encapsulation). Aceasta permite schimbări ale unei componente specifice fără afectarea altor componente;
- Interfață cu alte obiecte (interface sau messaging);

- Recunoașterea diferitelor tipuri de lucruri și comportamentul lor fără analiza lor;
- Folosirea aceleiași cereri pentru a invoca diferite implementări ale aceleiași acțiuni pentru două obiecte diferite (polymorphism);

1.6 Sisteme de gestiune a bazelor de date relationale în Oracle

Oracle furnizează un SGBDR flexibil - Oracle Database. Trăsăturile acestui SGBDR permit stocarea și gestiunea datelor cu toate avantajele structurilor relationale plus PL/SQL, un motor care permite stocarea și executarea unităților de program. Serverul permite utilizatorilor opțiunea de extragere de date bazată pe tehnici de optimizare. Sunt incluse trăsături de control a modului de accesare și folosire a datelor. Alte caracteristici sunt consistența și protecția prin mecanisme de blocare.



Oracle Corporation este la ora actuală cea mai mare companie de software din lume care are produsele axate pe baze de date. Este o suprematie câștigată de aproximativ 20 de ani în fața principalului concurrent, IBM – firma care a dezvoltat primul sistem de management al bazelor de date, numit System R, în anii 1970. Lansarea produsului Oracle a reprezentat startul în domeniul bazelor de date relationale comerciale construite pe principiile fundamentate matematic de către E.F.Codd. Software-ul care constituie baza de integrare a tuturor celorlalte produse ale companiei este Oracle Server – serverul de baze de date ajuns acum la versiunea 10 (Oracle Database 10g Release 2 (10.2.0.1.0)), disponibil pe mai multe sisteme de operare și platforme hardware: Microsoft Windows pe 32 de biți și 64 de biți, Linux x86 pe 32 de biți

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

și 64 de biți, Linux Itanium, HP-UX PA-RISC (64 biți), Solaris SPARC (64 biți), AIX 5L pe 64 biți, etc.

Succesul firmei Oracle s-a bazat în primul rând pe serverul de baze de date, pe baza căruia s-au dezvoltat de-a lungul timpului suita de dezvoltare Oracle Developer Suite (care cuprinde Oracle Forms, Reports, Designer, JDeveloper) pentru proiectarea de aplicații, Warehouse Builder și Discoverer pentru domeniul numit Business Intelligence, serverul de aplicații bazat pe J2EE – Oracle Application Server și suita back-office Oracle Collaboration Suite. În principiu toate aceste aplicații sătélit țin pasul, ca denumire de versiune, cu versiunea bazei de date.

Serverul de baze de date Oracle a căpătat de-a lungul timpului îmbunătățiri care au menținut acest produs pe primul loc. Ca istoric, se poate considera *versiunea 2* ca fiind prima versiune completă bazată pe caracteristicile relaționale introduse de limbajul SQL.

Versiunea 3 se caracterizează prin rescrierea în întregime a versiunii anterioare în limbajul C pentru asigurarea portabilității pe mai multe platforme. O altă caracteristică a versiunii 3 o reprezintă respectarea principiului atomicității frazelor SQL în cadrul tranzacțiilor.

Versiunea 4 aduce ca noutate principiul consistenței datelor la citire, adică datele supuse prelucrărilor sunt intangibile celorlalte tranzacții.

Versiunea 5 aduce îmbunătățiri în partea de client / server, aducând o răspândire a produsului bazat pe folosirea bazei de date prin accesarea de pe terminale sau stații PC mai slabe, investiția majoră fiind necesară doar pentru server.

Versiunea 6 îmbunătășește mecanismele de blocare care, începând cu această versiune, se face la nivel de linie permitând ca o tranzacție să nu blocheze o tabelă întreagă, ci numai liniile din tabelă care sunt supuse prelucrării. În versiunea 7 se introduc procedurile stocate și declanșatoarele (triggeri în engleză).

Odată cu *versiunea 7* a apărut și varianta pentru calculatoare personale: ORACLE Personal Edition 7.2, care permite dezvoltarea aplicațiilor fără a fi necesară conectarea la un server în rețea, ceea ce a crescut mult popularitatea sistemului.

Versiunea 8 a introdus partea obiectuală derivată din principiile programării orientate pe obiect ca o necesitate a creșterii complexității structurii datelor. Practic, începând cu această versiune, Oracle a devenit o bază de date obiectual-relațională, fiind permisă stocarea în baza de date atât a structurilor tabelare cât și a obiectelor folosind metoda mapării. Versiunea Oracle 8i este o continuare a versiunii 8, la care se adaugă un mediu RunTime Java direct în motorul bazei de date pentru structurarea logicii procedurilor stocate și în limbajul Java, nu numai în PL/SQL (limbajul procedural Oracle introdus din versiunea 7). *i*-ul de la 8i vine de la aplicațiile bazate pe noile tehnologii ale internetului. Tot în acest context s-a introdus suportul pentru limbajul SQLJ care

permite ca instrucțiunile SQL de acces la date sa fie incluse în proceduri sau metode scrise în Java rezidente pe server sau pe stațiile client.

Versiunea 9i a îmbunătățit mecanismele de securitate a bazelor de date prin introducerea conceptului de bază de date virtuală, s-au adăugat facilități de stocare și accesare a documentelor XML (eXtensible Markup Language), îmbunătățind astfel limbajul SQL, îmbunătățirea scalabilității prin introducerea arhitecturii RAC (Real Application Cluster). Această arhitectură permite rularea unui singur server de baze de date pe mai multe calculatoare independente (noduri) care formează un complex, permitând recuperarea transparentă a căderilor oricărui nod din rețea.

Versiunea 10g aduce ca noutate utilizarea tehnologiei grid (g-ul vine de la **grid**). Astfel s-a introdus stocarea după principiul grid prin tehnologia ASM (Automatic Storage Management) care elimină necesitatea unui sistem de fișiere. Datele pot fi stocate pe calculatoare diferite, gestionate de sisteme de operare diferite, având în comun faptul că ele sunt legate între ele în rețea și au puse la dispoziție capacitatea de stocare și procesare în ASM. Dacă un disc este adăugat sau scos, ASM redistribuie datele în mod automat. Practic, conceptul de grid este superior conceptului de cluster, grid-ul bazându-se practic pe cluster. Așa cum s-a procedat cu resursele de stocare la fel se utilizează în comun și resursele de procesare, aplicându-se principiile echilibrării încărcării (load balancing în engleză).

1.7 Sisteme de gestiune a bazelor de date relationale în FoxPro.

Apărând ca urmaș a lui DBASE III și dotat astfel încât să satisfacă cerințele unui SGBD modern, cu mediu de dezvoltare integrat, FoxPro are una din cele mai atractive interfețe utilizator și un compilator foarte puternic.

Versiunea 1.02 (anul 1989)

- sintaxa se bazează pe Foxbase (și deci pe dBase III), dar a fost extinsă cu multe instrucțiuni și funcții utile;
- mediu de programare confortabil, cu editoare integrate, operare cu mouse-ul, prelucrarea simultană a mai multor fișiere;
- depanator (debugger) și program de testare (trace) integrate în mediul de dezvoltare, precum și generatoare de machete, de programe și program de documentare automată a surselor;
- compilatorul nu generează cod executabil complet (.EXE) ci se obține un format intermedian care necesită existența mediului FoxPro sau a versiunii RunTime (viteză sporită).

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Versiunea 2.0 (anul 1991)

- compatibilitate sporită cu DBase IV (modul SQL incorporat, indecsă compusă);
- instrumente specifice de proiectare (generatoare de meniuri, rapoarte, aplicații);
- utilizarea tehnologiei de optimizare Rushmore;
- control superior al programatorului asupra evenimentelor din timpul rulării;
- help sensitiv la context;
- rulează pe rețele de calculatoare, în regim multiutilizator;
- este furnizată în două variante: standard (pentru mașini cu procesor 8088 și 80286) și extinsă (pentru mașini cu procesor 80386 și mai puternice).

Versiunea 2.5 pentru DOS (anul 1993, după preluarea de către Microsoft)

- versiune îmbunătățită și mult mai rapidă;
- optimizare Rushmore îmbunătățită;
- import / export date în formate Excel, Paradox;
- este furnizată de asemenea în două variante.

Versiunea 2.5 pentru WINDOWS (anul 1993)

- facilități specifice mediului Windows: Dynamic Data Exchange (DDE), Object Linking and Embedding (OLE);
- îmbunătățirea interfeței cu utilizatorul;
- facilități multitasking;
- controlul centralizat al perifericelor;
- sistem Help performant.

Versiunea 2.6 pentru DOS (anul 1994)

- mecanisme de convertire automată a elementelor DBase în elemente FoxProW;
- module de asistență a utilizatorului (Table/Raport/Screen/Query/Label Wizard);
- prezența unui Catalog Manager;
- varianta profesională permite conectivitate client-server.

Visual FoxPro 3.0 (anul 1995)

- utilizarea principiilor programării orientate obiect (POO);
- mediu orientat pe evenimente (programarea orientată eveniment);

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

- apropierea de modelul relațional al bazelor de date (introducerea dicționarului de date);
- extensii orientate obiect ale limbajului de programare;
- extinderea instrumentelor de proiectare și a numărului de asistenți;
- dezvoltarea rapidă a aplicațiilor prin gestionarul de aplicații Project Manager;
- apropierea FoxPro-Acces.

Visual FoxPro 5.0 (anul 1997)

- simplificarea accesului la fișierele de date sub alt format (via **Open DataBase Connectivity - ODBC**);
- îmbunătățirea mediului client-server;
- facilități web.

Visual FoxPro 6.0 (anul 2000)

- îmbunătățirea Class Browser-ului pentru eficiență programării;
- noi facilități în programarea orientată-obiect și web;
- adăugarea de noi metode și proprietăți.

Visual FoxPro 7.0 (anul 2002)

- editorul Visual FoxPro este îmbunătățit cu tehnologia IntelliSense (autocompleteare pe măsură ce se scrie cod);
- servicii web – ca și obiecte (clase);
- suportă XML;
- accesul datelor prin providerul OLE DB.

Visual FoxPro 8.0 (anul 2003)

- îmbunătățiri ale IDE (Interactive Development Environment): Task Pane Manager, Toolbox, IntelliSense pentru fereastra Watch, etc.;
- îmbunătățiri ale limbajului de programare (clase, comenzi și funcții noi);
- introducerea noii clase XMLAdapter și clase XML Web Services;
- conectivitate la date aflate la distanță folosind clasa CursorAdapter;
- introducerea tipului autoincrement;
- inserarea rândurilor prin comanda SQL Select.

Visual FoxPro 9.0 SP2 Sedna CTP (anii 2004 - 2008)

- ultima versiune – anunțându-se stoparea dezvoltării produsului de către Microsoft, acesta convergând de-a lungul ultimilor 8 ani către Microsoft SQL Server;
- soluție compatibilă .NET cuprinzând și servicii Web XML;

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

- compatibilitate și interoperabilitate mărită cu SQL Server pentru a facilita schimbul de date între cele două produse (introducerea noilor tipuri de date Varchar, Varbinary și Blob);
- introducerea funcției CAST() pentru conversia între tipuri de date;
- introducerea funcției inline ICASE() similară cu instrucțiunea DO CASE;
- îmbunătățirea raportării;
- îmbunătățiri la limbajul SQL (eliminarea limitării la 9 clauze de join și subinterrogări în comanda SQL Select);
- eliminarea limitărilor pentru dimensiunea șirurilor (array) și lungimea procedurilor;
- Tolbox-ul este dock-abil către Desktop sau alte ferestre IDE (în special s-a lucrat la dock-abilitate și la clase și funcții).

Avantaje în utilizarea FoxPro:

- tehnici de optimizare performante;
- limbaj performant de descriere și manipulare a datelor;
- încorporarea unui modul SQL foarte rapid;
- mediu favorabil elaborării programelor puternic modularizate și structurate, orientate obiect;
- mecanisme evolute de conversie a programelor și a datelor (DBase-FoxPro), suport bine pus la punct pentru XML;
- utilizarea în rețele de microcalculatoare, ca și client COM și COM server ;
- instalarea modulară în funcție de necesități și posibilități;
- include instrumente de proiectare de tip 4GL (**4-th Generation Language**);

Dezavantajele sistemului FoxPro:

- relativă simplitate a mediului (nu se pot elabora aplicații foarte complexe);
- nu este un sistem relațional pur (de exemplu, lipsa dicționarului de date la versiunile mai mici de 3.0);
- nu prezintă intrinsec caracteristici client-server, prezintă dezavantaje în rol de server într-o arhitectură client-server;
- nu este recomandat pentru WAN (**Wide Area Network**);
- nu prezintă modalități avansate de protecție și securitate a datelor;
- compatibilitatea versiunilor nu este asigurată în toate cazurile;
- viteza de execuție este variabilă (influențată de mulți factori).

Capitolul 2 Concepte și elemente de baza în FoxPro.

Sistemul FoxPro se adresează, în principal, către două categorii de utilizatori :

- proiectanții și programatorii care scriu aplicații în FoxPro.
- utilizatorii finali care își pot administra cu ușurință datele, folosind interfața utilizator;

FoxPro lucrează cu următoarele elemente de bază:

- **fișier de date**: colecție de informații organizate matricial;
- **fișier index**: colecție de informații auxiliare asociate unui fișier de date cu rolul de a permite parcurgerea acestuia într-o anumita ordine, respectiv pentru regăsirea rapidă a informațiilor pe baza unor coduri;
- **machetă (ecran, form)**: interfață de afișare și/sau de introducere a informațiilor într-un fișier de date;
- **raport** : forma de afișare a informațiilor dintr-un fișier de date la ecran, imprimantă sau fișier, având la baza principiul benzilor;
- **interrogare**: fișier de date virtual, obținut din unul sau mai multe fișiere de date fizice, interconectate între ele pe baza unor relații;
- **program**: colecție de instrucțiuni FoxPro ordonate conform unui algoritm ce urmărește, în general, prelucrarea informațiilor din unul sau mai multe fișiere;
- **aplicație**: program special ce integrează elementele FoxPro care au ca scop realizarea unui grup de operații unitare;
- **proiect**: fișier de date special ce integrează elementele FoxPro care au ca scop realizarea unui grup de operații unitare și care se folosește pentru generarea de aplicații și programe (tip EXE).

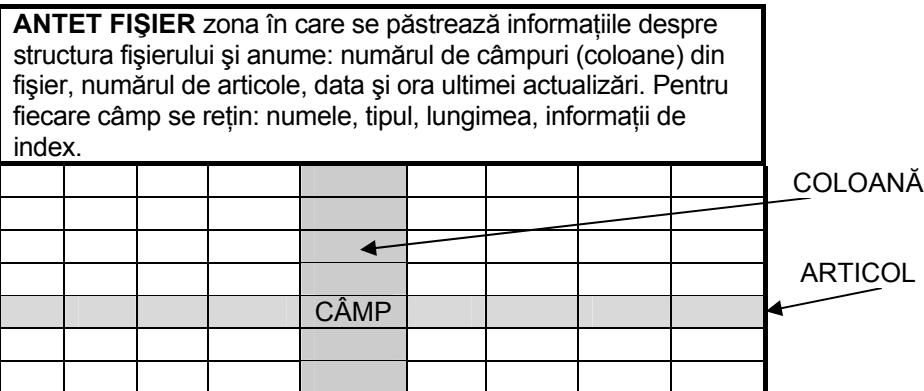
În continuare sunt prezentate sumar aceste elemente de bază ale mediului FoxPro, urmând ca pe parcursul celorlalte laboratoare să se prezinte instrucțiunile și modalitățile de lucru cu aceste elemente.



Fișier de date - reprezintă conceptul central al sistemului FoxPro

Fișierele de date FoxPro au o organizare specială ilustrată în figura următoare.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



Se disting două părți importante în acest fișier: zona propriu-zisă de informații împărțită în rânduri numite articole și coloane de câmpuri și antet fișier în care se păstrează informațiile de organizare a zonei propriu-zise.



ARTICOL: porțiune elementară a fișierului la care utilizatorul are acces la un moment dat; articolul este format din mai multe câmpuri.

CÂMP: unitatea elementară de memorare în fișier ce se regăsește la intersecția dintre un articol și o coloană.

COLOANA: toate câmpurile dintr-un fișier ce se caracterizează prin același nume, tip, lungime și se află în articole diferite.

Fișier index

Unui fișier de date FoxPro î se pot asocia unul sau mai multe fișiere index cu scopul de a permite regăsirea rapidă a articolelor în funcție de cheile realizate pe baza unor câmpuri sau combinații de câmpuri.

Organizarea fișierului index este prezentată în figura următoare:

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

ANTET INDEX	
	1
	2
	i`
	n`

Valoare cheie (ordonată) Număr de ordine articol în fișier

Valoare cheie: valorile câmpurilor sau combinațiilor de câmpuri din fișierul de date ordonate crescător sau descrescător, după opțiune;

Nr. ordine articol în fișier: numerele de ordine ale articolelor din fișier.

Cu alte cuvinte un fișier index reprezintă perechi formate din valori și numere de articol pentru care s-au obținut aceste valori, prin evaluarea unei chei ordonate crescător sau descrescător.

Numărul de perechi n` este mai mic sau egal ca numărul de articole din fișierul de date asociat.

Avantajul căutării în fișierul de date printr-un fișier index asociat rezultă tocmai din faptul că valorile sunt ordonate, deci căutarea în index este relativ simplă și rapidă, după care, prin obținerea numărului de ordine al articolului căutat, căutarea în fișierul de date devine de asemenea simplă și ușoară.

În sistemul FoxPro, începând cu versiunea 2.0 s-a introdus un nou tip de fișier index și anume fișierul multiindex compact (CDX). Fișierul multiindex se aseamănă cu cel monoindex numai ca are mai multe perechi de coloane valoare cheie-nr. ordine articol în fișier. Perechile de coloane valoare cheie și nr. ordine articol au nume unic relativ la fișier și poartă numele de index (tag).

Macheta – Crearea și gestionarea unei machete are la bază un program special care realizează operații de afișare și/sau operații de întreținere a fișierelor de date, realizând astfel o interfață între utilizator și sistemul FoxPro.

Avantajul folosirii machetelor constă, în primul rând, în existența unui editor specializat în crearea și modificarea acestor machete într-un mod facil. De asemenea limbajul de programare dispune de instrucțiuni de lucru cu machete.

Raportul – reprezintă modalitatea prin care se pot realiza operații de afișare sau de tipărire a informațiilor din fișierele de date. Un raport conține mai multe benzi pentru o grupare lizibilă a informațiilor. Avantajul folosirii rapoartelor constă, în primul rând, în existența unui editor specializat în crearea și

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

modificarea acestora, în care este suficient să definim informațiile (text utilizator și câmpuri) care se vizualizează în fiecare banda pentru ca afișarea/tipărirea informațiilor să aibă loc în forma dorită.

Interrogarea (Query) – Reprezintă un fișier de date virtual ce înglobează date din mai multe fișiere fizice de date prin stabilirea de relații între câmpurile de același tip. Se permite de asemenea filtrarea informațiilor (câmpuri și/sau articole) din oricare din fișierele fizice. FoxPro dispune de un editor specializat în crearea și modificarea lor. Avantajul folosirii interogărilor constă în economia de spațiu necesară memorării unui fișier virtual față de unul fizic echivalent, precum și din întreținerea ușoară a elementelor componente.

Programul – reprezintă unitatea elementara în care se pot grupa instrucțiunile, conform unui algoritm. La nivelul sistemului, programele se întrețin printr-un editor de texte încorporat.

Aplicația – ca program integrator de elemente FoxPro, se poate realiza și pe căi tradiționale folosind editorul de programe și scriind efectiv un astfel de program. Metoda aceasta este specifică majorității sistemelor de dezvoltare programe în diferite limbi de nivel înalt (B. Pascal, B C++ etc.). Față de acestea, sistemul FoxPro dispune de un editor specializat în proiectarea de programe prin integrarea diferitelor elemente FoxPro, bazat pe noțiunea de proiect.

Proiectul – este un fișier de date special, integrator de obiecte FoxPro, care se folosește pentru generarea de aplicații sau programe executabile. Sistemul FoxPro este capabil să creeze două tipuri de formate executabile : primul numit “compact”, care necesită existența unor biblioteci FoxPro, iar al doilea numit “stand-alone”, care nu necesită nici un fișier suplimentar pentru a fi lansat în execuție.

2.1 Descrierea mediului integrat

FoxPro este un sistem ce dispune de un mediu integrat compus din modul de execuție, compilator încorporat în sens FoxPro, generator (pentru machete, interogări, meniuri, aplicații), interfață utilizator (formată din Meniul Sistem, Fereastra de Comanda, linia de stare), editoare, depanator simbolic la nivelul limbajului (Trace-Debug), modul SQL inclus, help, istoric al comenziilor cu posibilități de editare ca la un program (fereastra Command), alte utilitare (sistem de gestiune a fișierelor DOS-System, Filer, calculator, etc.).

2.2 Tipuri de date în FoxPro, operatori, funcții

Un tip de dată reprezintă o caracteristică a datelor prin care se stabilește ce operații se pot executa asupra lor, modul de codificare a datelor în memoria calculatorului, semnificația acestor date.

Tipurile de date acceptate de limbaj sunt: numeric, logic, dată calendaristică, sir de caractere, memo, general.

Pentru fiecare tip de dată se vor prezenta: modalitatea de specificare a tipului de dată, operatorii ce se aplică acestor date, comenzi și funcțiile referitoare la datele de tipul respectiv.

Pentru mai multe detalii asupra funcțiilor sau instrucțiunile prezentate se recomandă a se consulta Help-ul.

Tipul numeric – memorează numere pozitive/negative cu punct zecimal fix; lungimea câmpului numeric este limitată la 20 cifre (*semnul și punctul zecimal ocupă câte o cifră!*). Se folosește pentru a stoca valori numerice în câmpurile unui tabel, valorile numerice returnate de funcții, variabile de tip numeric, constante numerice. Se pot evidenția următoarele:

- **Tipul numeric simplu (Numeric sau Float)** – este tipul numeric clasic. Zona de memorie alocată pentru acest câmp este dependentă de lungimea declarată de utilizator la crearea tabelului.
- **Tipul numeric dublu (Double)** – datele de acest tip reprezintă numere memorate în virgula mobilă. Lungimea zonei de memorie ocupate este fixă, de 8 octeți, și se folosește pentru memorarea numerelor foarte mari.
- **Tipul numeric întreg (Integer)** – este specific numerelor întregi. Lungimea zonei de memorie ocupată este fixă și este de 4 octeți. Se preferă folosirea acestui tip de dată atunci când folosim valori întregi, mai mari de 9999.
- **Tipul monetar (Currency)** – se folosește la memorarea valorilor exprimate în bani. Zona de memorie alocată are 8 octeți. O valoare de acest tip se declară prin introducerea în fața valorii numerice a simbolului monetar (\$). Exemplu A=\$500.47

Operatori algebrici	
(,)	Grupează expresiile
** , ^	Ridicare la putere
* , / , %	Înmulțire, împărțire, modulo
+ , -	Adunare, scădere

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Operatori relationali	
<	Mai mic decât
>	Mai mare decât
=	Egal
<>, #, !=	Diferit de
<=	Mai mic sau egal cu
>=	Mai mare sau egal cu

Comanda de control pentru afișarea valorilor cu un anumit număr de zecimale este : SET DECIMALS TO

Efect: determină numărul minim de cifre zecimale ce sunt afișate în cazul în care nu se specifică un format explicit de afișare. *Implicit acest număr este 2.*

Exemplu :

```
? 5/11
SET DECIMALS TO 4;
? 5/11
```

Funcții aplicabile tipului de dată numeric

Funcții referitoare la semnul datelor numerice:	
ABS(<expn>)	returnează valoarea absolută a unei expresii numerice
SIGN(<expn >)	returnează un număr indicând semnul matematic al expresiei numerice transmise ca parametru și anume: +1 pentru numere > 0, -1 pentru cele mai mici ca zero sau 0
Funcții de aproximare a datelor numerice:	
INT(<expn >)	trunchiază rezultatul unei expresii numerice la partea întreagă
ROUND (< expN1>, <expN2>)	returnează numărul expN1 rotunjit la numărul de zecimale date de expN2. Dacă expN2 este negativ, se rotunjește la zero
CEILING(<expn >)	returnează cel mai apropiat întreg mai mare sau egal ca numărul expn
FLOOR(<expn >)	returnează cel mai apropiat întreg mai mic sau egal ca numărul expn

Functii matematice elementare:	
EXP(<expn >)	returnează rezultatul ridicării constantei "e" la puterea dată ca parametru
LOG(<expn >)	returnează logaritmul natural / zecimal pentru numărul specificat
SQRT(<expn >)	returnează valoarea radicalului din numărul (pozitiv) specificat
MOD (<expN1>, <expN2>)	returnează un întreg, restul împărțirii <expN1> la <expN2>. Semnul rezultatului este cel al <expN2>.
MAX (< exp1>, < exp2>, < exp3>) sau MIN(< exp1>, < exp2>, < exp3>)	returnează maximul / minimul dintre două expresii numerice
RAND (< expN>)	returnează un număr aleator în intervalul 0 și 0.999999. Fără parametru sau cu o valoarea negativă se folosește pentru inițializarea ceasului sistem. Implicit este 0.1000001.

Tipul logic – are un singur octet lungime și poate memora două valori: ‘adevărat’ (.T.) True și ‘fals’ (.F.) False; valoarea nulă este tratată de FoxPro ca fiind .F. Se folosește pentru a stoca valori în câmpurile de tip logic ale tabelelor din baze de date, pentru a stoca valorile returnate de funcții, variabile de tip logic sau alte expresii logice.

Operator logic	
(,)	Grupează expresiile
!, NOT	Negație logică
AND	ȘI logic
OR	SAU logic

Exemplu :



? NOT (1=3)
.T.
? (1<=4) AND (5>3)
.T.
? 6<3 OR 4*2=9
.F.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Tipul dată calendaristică – are 8 octeți lungime; datele de acest tip sunt memorate intern de FoxPro ca cifre ASCII în format AAAALLZZ, ceea ce permite efectuarea de calcule aritmetice cu ele. Implicit FoxPro nu permite introducerea de date inexistente în fișierele de date. FoxPro tratează datele calendaristice invalide ca date calendaristice vide.



Formatul de afișare a datelor calendaristice este controlat de comanda:

SET DATE [TO] american | ansi | british | french | german | italian | mdy | dmy | ymd

Tipul de dată calendaristică	Formatul de afișare
AMERICAN	LL/ZZ/AA
ANSI	AA.LL.ZZ
BRITISH	ZZ/LL/AA
FRENCH	ZZ/LL/AA
GERMAN	ZZ.LL.AA
ITALIAN	ZZ-LL-AA
JAPAN	AA/LL/ZZ
USA	LL-ZZ-AA
MDY	LL/ZZ/AA
DMY	ZZ/LL/AA
YMD	AA/LL/ZZ

Exemplu:



```
store {"^2006-06-21} to data  
? DATA  
06/21/2006  
SET DATE BRIT  
? DATA  
21/06/2006
```

SET CENTURY ON | OFF

Efect: ON – se afișează anul pe 2 cifre;
OFF – se afișează anul pe 4 cifre

SET MARK TO [<c>]

Efect: Stabilește caracterul delimitator.

Functii pentru tipul de data calendaristica	
DATE()	obtinerea datei curente a sistemului
DOW(<eD>)	returnează valoarea numerică a zilei săptămânii dintr-o expresie de tip dată calendaristică (începe cu 1 pentru duminică)
CDOW(<eD>)	returnează un sir de caractere ce reprezintă numele zilei din cadrul săptămânii pentru data specificată
DAY(<eD>)	returnează numărul zilei din cadrul lunii pentru data specificată;
MONTH(<eD>)	returnează numărul lunii pentru data specificată
CMONTH(<eD>)	returnează un sir de caractere ce reprezintă numele lunii din cadrul unei expresii de tip dată
YEAR(<eD>)	returnează valoarea numerică a anului pentru data specificată
DMY (<expD>)	convertește o dată de intrare de tip calendaristic în formatul zi-luna-an (returnează un sir de caractere)
MDY (<expC>)	convertește o dată de intrare tip calendaristic în formatul lună-zi-an (returnează un sir de caractere)
Functii pentru controlul timpului	
TIME(<eN>)	returnează un sir de 8 caractere în formatul HH:MM:SS ce reprezintă ora exactă a sistemului

Instrucțiuni pentru controlul formatului de afișare al ceasului sistem:

SET CLOCK ON | OFF
SET CLOCK TO [<ROW>,<COL>]
SET HOURS TO [12 | 24]

Tipul de dată caracter – permite memorarea oricărui caracter ASCII, limita impusă fiind de 254 caractere. Sirul de caractere este o mulțime ordonată de caractere ce se tratează ca un tot unitar. Fiecare caracter i se poate asocia un număr reprezentând poziția acestuia în cadrul sirului (primul caracter are poziția 1).



Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Se folosește pentru a stoca valori în câmpurile de acest tip ale tabelelor din baza de date, valorile returnate de funcții pentru date calendaristice, variabile de tip dată calendaristica, constante de tip dată calendaristica.

Operatori de concatenare	
+	concatenează șiruri, eventualele spații finale ale fiecărui șir rămânând între șiruri
-	concatenează șiruri, spațiile finale fiind duse la sfârșitul șirului rezultat

Operatori de comparare	
<, >, =	mai mic, mai mare, egal
<>, #	diferit de
<= sau <=	mai mic sau egal
>= sau >=	mai mare sau egal
\$	folosit pentru concatenarea șirurilor de caractere
==	identic

Observații: Operatorii de comparare returnează un rezultat adevărat (.T.) sau fals (.F.) putând fi utilizati în compararea a două expresii de același tip, indiferent de tipul lor. Compararea a 2 șiruri de caractere se face prin compararea codurilor ASCII ale caracterelor de pe pozițiile echivalente.

Exemplu :



expresia :‘strada_ +’George _Cosbuc’ după evaluare devine :

‘strada_George_Cosbuc’

expresia ‘Salut - prieteni !’ după evaluare devine: ‘Salut prieteni! ‘

Funcții pentru tipul de dată caracter	
LEN (<expC>, <memo field>)	returnează o valoare numerică ce indică numărul de caractere dintr-o expresie de tip caracter sau dintr-un câmp memo
AT (<expC>,<expC>/<memo field>)	returnează un număr ce indică poziția de la care începe primul șir de caractere ca subșir al celui de-al doilea. Indicele primului caracter dintr-un șir este 1.
LEFT (<expC>,<mem	returnează un șir de caractere ce indică primele <expN> caractere din <expC>

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

<i>ield>, <expN></i>	- dacă <expN> este zero, returnează sirul vid - dacă <expN> este mai mare decât <expC>, se returnează <expC> în întregime
<i>RIGHT (<expC>,<memo field>, <expN>)</i>	extragă și returnează un sir de caractere compus din ultimele <expN> caractere ale <expC> - dacă <expN> este zero sau mai mic ca zero, returnează sirul vid - dacă <expN> este mai mare decât <expC>, se returnează <expC> în întregime
<i>SUBSTR (<expC>,<memo field>, <start position>,<number of characters>)</i>	extragă și returnează din expC un subșir de caractere ce începe în poziția specificată <start position> de lungimea indicată de <number of characters>
<i>LOWER(<expC>)</i>	convertește literele mari în litere mici
<i>UPPER(<expC>)</i>	convertește literele mici în litere mari
<i>LTRIM (<expC>)/RTRIM (<expC>)</i>	șterge spațiile de la stânga/ dreapta unui sir de caractere și returnează rezultatul
<i>TRIM (<expC>)</i>	șterge spațiile de la sfârșitul unui sir de caractere și returnează rezultatul
<i>CHR (<expC>)</i>	convertește o expresie numerică într-o caracter
<i>MAX (<expC1>,<expC2>)</i>	returnează maximul/minimul dintre două expresii de tip caracter
<i>MIN (<expC1>,<expC2>)</i>	
<i>SPACE (<expN>)</i>	generează un sir de caractere compus din <expN> spații
<i>LIKE (<pattern>,<expC2>)</i>	compara cele două siruri de caractere returnând .T. sau .F.



Exemplu :

? LEN ('ACEST sir ARE 26 CARACTERE')
? AT ('P','IN CE POZITIE ESTE LITERA P ? ')
? AT ('AB', 'ABAC,ABACD',2)
? AT ('AB', 'ABAC,ABACD,ABCDE',3)
? LEFT ('CALCULATOR',3)
? RIGHT ('CALCULATOR',3)
? SUBSTR('CALCULATOR',4,3)

? LOWER('AAAAAAAAAAAAAAA')

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

? UPPER('bbbbbbbb')

STORE 'casetofon' TO c1

STORE 'portabil' TO c2

? c1,c2

? c1, LTRIM(c2)

USE AG

? 'SALARIUL D-LUI(D-NEI) '+ RTRIM(ume) + ' ESTE ' +
LTRIM(str(sal_baza))

? CHR(65)

? MAX(54, 39, 40)

? MAX(2^2, 10*12, 2*3)

? MAX({31/01/2001},DATE())

? MAX('AAA', 'BBB', 'ABC')

? MAX('AAA', 'BBB', 'CCC')

? LIKE ('CARACTER', 'AR')

? LIKE ('RA', 'AR')

? LIKE ('AR', 'AR')

Tipul memo – permite memorarea cantităților mari de text (fără limitarea la 254 caractere impusă pentru datele de tip Caracter). De asemenea permite memorarea de fișiere executabile, imagini, sunet. Singura limitare este dată de dimensiunea spațiului liber pe disc.



Utilizarea câmpurilor memo se recomandă în situațiile în care apar diverse comentarii și, evident, nu în toate înregistrările (ar fi ineficient să se stabilească un câmp caracter cu lungimea 70 doar pentru explicațiile care apar în câteva înregistrări). Informațiile din câmpul memo nu sunt memorate în același fișier cu restul câmpurilor.

O tabelă ce conține un câmp memo are asociat un fișier cu extensia FPT numai pentru datele memo.

Editarea repetată a datelor memo conduce la fragmentări și se recomandă utilizarea comenzi PACK sau PACK MEMO ce determină economie de spațiu pe disc și îmbunătățește viteza de depozitare a datelor.

Pentru stabilirea dimensiunii blocurilor dintr-un câmp memo se poate folosi comanda SET BLOCKSIZE. Majoritatea programatorilor folosesc dimensiunea minima de 33 octeți (valoarea implicită este de 64 octeți) pentru un

bloc, care asigură o eficiență mai mare în ceea ce privește dimensiunea fișierului memo.

Tipul de dată general – este utilizat pentru legarea și înglobarea obiectelor (OLE) cum ar fi desene, documente, sunete, grafică.



Observație: Prioritatea în cazul folosirii într-o expresie a mai multor operatori este următoarea :



1. operatori matematici și pe siruri de caractere
2. operatori de comparare
3. operatori logici
4. toate operațiile ce sunt pe același nivel se execută de la stânga la dreapta.

Pentru modificarea ordinii operațiilor se pot folosi parantezele rotunde.

2.3 Funcții pentru conversii între tipuri de date

Din număr în sir de caractere	
STR (<expN>[,<lenght>][,<decimal>])	convertește un număr expN într-un sir de caractere, returnând un sir de caractere. <lenght> indică numărul total de caractere al sirului creat, iar <decimal> indică numărul de zecimale
Din sir de caractere în număr	
VAL (<expC>)	convertește un sir de caractere într-un număr, returnând rezultatul. Numărul de zecimale este cel dat de SET DECIMALS
Din sir de caractere în dată	
CTOD (<expC>)/{ expC)	convertește o dată de intrare tip sir de caractere (delimitate ca sir de caractere) într-o variabilă de tip dată calendaristică, în formatul setat prin SET DATE

Din dată în sir de caractere	
DTOC (<expC>)/{ expC)	convertește o dată de intrare de tip calendaristic în sirul de caractere corespunzător mm/dd/yy

Exemple:



```
? STR(25.6, 4, 1)
? STR(25.6, 4, 0)
STORE '12' TO A
STORE '13' TO B
? VAL(A) + VAL(B)
```

2.4 Zone de lucru

FoxPro folosește aşa numitele “zone de lucru” (în număr de 255) pentru manipularea fișierelor de date. Identificarea zonelor de lucru se face cu numere de la 1 la 255 (pentru primele 10 zone se pot folosi literele A..J). Numai o zona de lucru din cele 255 este curentă la un moment dat. Instrucțiunile referitoare la fișierele de date lansate la un moment dat vor acționa asupra fișierului deschis în zona curentă activă. La deschiderea unui fișier de date într-o zona de lucru, acestuia i se atribuie un nume numit “alias” (pseudonim).

Comanda pentru selectarea unei zone de lucru are sintaxa:

SELECT <eN>|<eC>

unde :<eN> = numărul zonei de lucru activate;

<eC> = sir de caractere ce reprezintă aliasul tablei.

Observații: dacă eN=0 se selectează prima zonă de lucru neocupată.

Deschiderea unui fișier (tabel) de date se face folosind comanda :



```
USE [<file>|?] [IN <expN1>] [AGAIN]
[INDEX <index file list>]
[ORDER [<expN2>|<idx index file>] [TAG]<tag name> OF<cdx>]
[ASCENDING|DESCENDING]]]
[ALIAS <alias>] [EXCLUSIVE] [SHARED] [NOUPDATE]
```

Clauze:

AGAIN – deschiderea acelaiași fișier în mai multe zone de lucru;

NOUPDATE – nu permite modificarea datelor (structură și conținut);

SHARE, EXCLUSIVE – pentru lucru în rețea;

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

ALIAS – aliasul atribuit de utilizator fișierului de date.

Prin specificarea numelui fișierului de date acesta se deschide în zona de lucru curentă (dacă nu se specifică zona de lucru).

Dacă se deschide un fișier de date într-o zona de lucru în care s-a deschis anterior un alt fișier, atunci fișierul anterior deschis se închide automat.

Închiderea unui fișier (tabel) de date se face folosind comanda :

USE

Deschiderea unei baze de date se face folosind comanda :

OPEN DATABASE [<file> | ?]

Închiderea unei baze de date se face folosind comanda :

CLOSE DATABASES (închide toate fișierele de date și index asociate deschise)

CLOSE ALL (închide și ferestrele activate anterior)

2.5 Construirea bazelor de date relaționale. Componente.



O bază de date relațională reprezintă o structură folosită la memorarea și gestionarea datelor, descriind un anumit tip de obiecte.

Ea are asociat un fișier special numit *dicționar de date* (cu extensia dbc) în care sunt memorate date referitoare la bază în ansamblul sau, cum ar fi: tabelele componente, relațiile permanente între tabele, procedurile stocate, etc. Un program care prelucreză o bază de date relațională citește mai întâi *dicționarul de date* datele și apoi, cunoscându-i structura complexă, procedează la prelucrarea datelor respective.

În Visual Fox Pro termenii de "baza de date" și "tabela" nu sunt sinonimi. Termenul de bază de date (fisierdbc) se referă la o bază de date relațională, care este un container cu informații despre unul sau mai multe tabele (fisierdbf) sau vederi.

Construirea unei tabele simple



**CREATE TABLE|DBF dbf_name>
(<fname1><type>[(<precision>[,<scale>])[,<fname2>...]])**

Clauze:

<dbf_name> = numele tabelei create (tabelul este deschis într-o zona de lucru nouă

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

<fname1>, <fname2> = numele câmpurilor din noul tabel creat;
<type> - un caracter ce indica tipul datei pentru câmpul respectiv;

<type>	<precision>	<scale>	Descriere
C	n	-	Șir de caractere de lungime n
D	-	-	Data calendaristică
N	n	d	Numeric, de lățime n cu d zecimale
L	-	-	Logic
M	-	-	Memo



ALTE COMENZI

Modificarea structurii unei tabele de date active se face prin comanda:

MODIFY STRUCTURE

Vizualizarea structurii unei baze de date:

DISPLAY STRUCTURE [IN <expN>|<expC>][TO PRINTER [PROMPT]]

LIST STRUCTURE[NOCONSOLE][TO PRINTER |TO FILE<file>]

O alta metodă de creare a fișierelor de date din fișiere înrudite o reprezintă comanda:

COPY STRUCTURE TO <file> [FIELDS <field list> [WITH CDX]

prin care se realizează copierea structurii (eventual parțiale) a unei tabele de date în structura alteia.

O alta modalitate o constituie folosirea unui fișier intermedian, folosind comenziile:

COPY STRUCTURE EXTENDED TO <file> [FIELDS <field list> | FIELDS LIKE <skel> | FIELDS EXCEPT <skel>]

și respectiv:

CREATE [<file1>] FROM [<file2>]

Pentru detalii urmăriți Help-ul.



Crearea unei baze de date

Pentru a crea o baza de date folosiți comanda:

CREATE database <dbc_name>

sau, din meniul File, alegeți opțiunea New și Database, după care apăsați butonul New File, ceea ce conduce la afișarea pe ecran a unei ferestre în care

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

trebuie specificat numele noii baze de date și calea în care aceasta va fi salvată. Pe ecran este deschisă fereastra Constructorului de baze de date (Database Designer), iar la meniul sistem se adăuga un nou submeniu, numit Database, care conține opțiunile necesare realizării operațiilor specifice, cum ar fi :

New table _ se creează o nouă tabelă în baza de date

Add Table _ se adăuga la bază o nouă tabelă existentă pe disc

O bază de date relațională include:

- mai multe tabele cu proprietățile descrise anterior;
- relațiile permanente dintre tabele, relații ce nu vor fi sterse odată cu terminarea programelor de prelucrare în care au fost definite (cum este cazul relațiilor temporare între tabelele);
- datele referitoare la tabele și relațiile dintre ele sunt memorate în dicționarul bazei de date și sunt restabilești automat la deschiderea acesteia;
- vederi, care reprezintă tabele virtuale construite pe baza uneia sau mai multor câmpuri din mai multe tabele;
- proceduri și funcții asociate bazei de date, care pot fi apelate în secvențele de cod asociate tabelelor componente;
- conexiuni, care reprezintă mecanismul de accesare a datelor din alte sisteme.

Caracteristicile tabelelor relaționate

Numele lungi pentru tabele și câmpurile acestora - Unei tabele îi se poate atașa un nume lung de maxim 128 caractere, care conduce la o mai bună lizibilitate. Această regulă este valabilă și pentru numele câmpurilor dintr-o tabelă, cu următoarele observații:

- aceste nume nu trebuie să conțină spații libere.
- sistemul nu face diferență între litere mari și mici.
- dacă o tabelă a fost creată mai întâi ca fiind una simplă și apoi a fost inclusă într-o bază de date, odată definite numele lungi pentru câmpurile acesteia, cele vechi (de maxim 10 caractere) nu mai pot fi folosite.

Comentarii și observații referitoare la tabelele legate și la câmpurile componente

Deși comentarea tabelelor și a câmpurilor acestora într-o bază de date nu este una dintre cele mai puternice facilități, este bine ca ea să fie folosită pentru o mai ușoară documentare ulterioară sau pentru o mai bună înțelegere a modului de concepere.

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

Formatul de afișare și citire a câmpurilor tabelelor legate

Pentru fiecare câmp se poate specifica un format de afișare implicit, care să servească la formatarea datelor atunci când aceste vor fi afișate (de exemplu când se folosește comanda browse). Formatul de afișare reprezintă un sir de caractere format dintr-o serie de coduri, în funcție de care sistemul stabilește modul de prezentare a datelor respective. De exemplu, un format de afișare de tipul 999.99 desemnează o valoare numerică afișată cu 3 cifre în fața punctului zecimal și două după acesta (vezi clauza Picture).

Valorile隐式 ale câmpurilor

Pentru fiecare câmp al unei tabele se poate preciza o valoare implicită care să fie încărcată automat la adăugarea unei noi înregistrări. Valoarea respectivă trebuie de să fie de același tip ca și câmpul. Se poate preciza deasemeni și o expresie a cărei evaluare va conduce la valoarea ce se va încărca automat.

Validarea la nivel de câmp

Pentru un câmp al unei tabele legate se poate preciza o expresie logică care se evaluatează în momentul schimbării valorii câmpului. Dacă valoarea expresiei este adevărată, se consideră că valoarea introdusă este corectă, dacă nu se afișează un mesaj de eroare, care poate fi cel implicit al sistemului sau unul personal, definit de utilizator (de exemplu: "Regula de validare este încălcată"). Observații : Evaluarea condiției are loc doar în momentul în care se introduc sau modifică datele din câmpul respectiv.

Validarea la nivel de înregistrare

Pentru o tabelă ce aparține unei baze de date se pot defini reguli de validare la nivel de înregistrare, constând într-o expresie logică care va fi evaluată în cazul în care se adaugă noi înregistrări, se modifică cele existente sau se sterg înregistrări.

Dacă expresia returnează un rezultat adevărat, operația se consideră corectă, dacă nu, operația nu este acceptată și se generează un mesaj de eroare care poate fi cel implicit al sistemului sau cel definit de proiectantul bazei de date.

În expresia de validare se pot încorpora funcții utilizator sau secvențe de program destul de complexe.

Observație: Validarea la nivel de înregistrare se declanșează după cea la nivel de câmp, dar înaintea secvențelor de cod asociate evenimentelor de manipulare a tabelei.

Secvențe de cod asociate evenimentelor de manipulare a unei tabele legate

Manipularea unei tabele se reflectă prin operațiile de adăugare de noi înregistrări, modificarea celor existente sau stergerea unor înregistrări. Pentru

fiecare dintre aceste evenimente se pot specifica secvențe de cod care se execută la apariția evenimentului respectiv.

Observație : secvențele de cod asociate evenimentelor de manipulare a tabelei legate se declanșează după validările la nivel de câmp și la nivel de înregistrare.

2.6 Comenzi pentru vizualizarea și modificarea datelor din tabele



EDIT [FIELDS <field list>][<scope>][FOR <expL1>][WHILE <expL2>] [FONT <expC1> [, <expN1>]][STYLE <expC2>][FREEZE <field>] [KEY <expr1> [, <expr2>] [NOAPPEND] [NOCLEAR] [NODELETE] [NOEDIT] | NOMODIFY]

EDIT este comanda ce permite afișarea și modificarea conținutului unui articol al tabelei sau al unui fișier creat cu **CREATE QUERY/VIEW**

- dacă nu se indică un interval <scope> sau o condiție **FOR**, **WHILE** se pot parcurge toate articolele din tabelă;
- se trece de la un câmp la altul cu tasta **TAB**, se trece la precedentul sau următorul articol cu săgeata sus sau jos;
- închiderea ferestrei de editare se face cu **CTRL/END** pentru a salva toate modificările sau cu **ESC** pentru abandon;
- dacă nu s-a specificat **NOAPPEND**, se pot adăuga articole în tabela prin **CTRL/N**. La terminarea comenzi se revine la locul de unde a fost lansată;
- **NOMENU** împiedica accesul la linia *menu*;
- **NOEDIT** împiedica modificarea datelor;
- **NODELETE** împiedica marcarea pentru ștergere;
- **NOCLEAR** păstrează ultimul articol pe ecran după ce se ieșe din **EDIT**;
- <expN1> indică cu al cătelea articol să se înceapă editarea;



BROWSE [FIELDS <field list>][FONT <expC1> [, <expN1>]] [STYLE <expC2>][FOR <expL1>][FORMAT][FREEZE <field>][KEY <expr1>[, <expr2>]] [NOINIT][LOCK <expN2>] [NOAPPEND][NOCLEAR] [NODELETE] [NOEDIT] [NOMODIFY] [NOLINK][NOMENU][TITLE <expC4>][VALID [:F] <expL2>] [ERROR <expC5>][WHEN <expL3>] [WIDTH <expN5>] [[WINDOW <window name1>][IN [WINDOW] <window name2>] [COLOR SCHEME <expN6>] COLOR <color pair list>]

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

BROWSE este comanda ce permite modificarea sau adăugarea de articole într-o tabelă :

- dacă nu este deschisă nici o tabelă se solicită numele celei dorite;
- câmpurile calculate sunt accesibile doar la citire;
- afişarea se face sub forma de tablou, în ordinea câmpurilor din tabelă sau a celor specificate în **FIELDS**, clauzele **Font** și **Style** determinând tipul și dimensiunea literelor ;
- comanda este asemănătoare cu **EDIT**;
- **WIDTH** limitează dimensiunea coloanelor pentru câmpurile caracter;
- **FREEZE** permite menținerea cursorului în același câmp al tabelei ;
- **WINDOW** activează o fereastră în care se va afișa tabelul;
- **FIELDS** permite alegerea câmpurilor și a ordinii de afișare;



APPEND FROM <file> |? [FIELDS <field list> | FIELDS LIKE <skel> | FIELDS EXCEPT <skel>] [FOR <expL>][[TYPE] [DELIMITED [WITH TAB | WITH <delimiter>]

APPEND permite adăugarea unuia sau mai multor articole din fișierul sursă specificat la sfârșitul tabelei active.

- se pot adaugă date și din fișiere de alt tip, nu doar din DBF. Dacă sursa și destinația sunt DBF, se copie doar câmpurile comune celor două tabele ;
- dacă SET DELETED este OFF se copie și articolele marcate pentru ștergere (dar nu și marcajul) ;
- dacă un câmp din sursă este mai mare decât omologul sau din destinație, datele de tip caracter în exces se pierd iar cele numerice sunt înlocuite de asteriscuri;
- dacă structurile celor două fișiere nu corespund se afișează mesajul “ 0 row(s) inserted”. Coloanele ce nu apar în tabel sunt ignorate;
- dacă nu se indică TYPE se presupune ca este un fișier sursă tip dbf;
- se pot importa date din fișiere de mai multe tipuri;
- pentru o mai bună inserare a datelor se specifică, unde este cazul, delimitatorul.



INSERT [BEFORE][BLANK] sau APPEND [BEFORE][BLANK]

INSERT adaugă un nou articol în tabelă, în poziția de articol curentă fără opțiuni se lansează introducere datelor în tabelă

- funcționează ca **APPEND**, permitând adăugarea mai multor articole succesiv;
- încheierea introducerii datelor se face cu **CTRL/END**;
- **BEFORE** face ca inserarea să se facă înaintea articolului curent;
- **BLANK** inserează un articol gol;
- dacă **SET CARRY** este **ON** se copiează în articolul inserat conținutul articolului precedent;



EXPORT TO <file> [FIELDS <field list> | FIELDS LIKE <skel> | FIELDS EXCEPT <skel>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [NOOPTIMIZE] [TYPE] DIF|MOD|SYLK|WK1|WKS|WR1|WRK|XLS]

EXPORT transferă datele dintr-un fișier în altul, de alt tip (implicit fișierul se consideră a fi de tip.dbf)



DELETE [<scope>] [FOR <expL1>] [WHILE <expL2>] [NOOPTIMIZE]

DELETE marchează pentru ștergere articolele specificate din tabel. Implicit se marchează articolul curent.

- ștergerea este logică, articolele putând fi readuse cu **RECALL**
- ștergerea fizică se face cu comanda **PACK**
- articolele marcate pentru ștergere sunt marcate cu o steluță în prima poziție la folosirea comenziilor **DISPLAY**, **LIST**



RECALL [<scope>] [FOR <expL1>] [WHILE <expL2>] [NOOPTIMIZE]

RECALL anulează marcajul pentru ștergerea articolelor specificate din tabel. Dacă articolele au fost șterse cu **ZAP** sau **PACK**, nu mai pot fi refăcute.



PACK

PACK șterge fizic articolele marcate pentru ștergere. Toate fișierele index deschise sunt automat reindexate.



ZAP

ZAP elimină toate articolele din tabelă. Toate fișierele index deschise sunt automat reindexate.

SET SAFETY ON / OFF

Dacă este **ON**, implicit se previne pierderea articolelor dacă se comanda crearea unui fișier cu același nume cu unul deja existent sau dacă se dă **ZAP**. Se cere un mesaj de confirmare pentru ștergere.

SET DELETED ON / OFF

Determină dacă articolele marcate pentru ștergere sunt sau nu luate în considerare de către alte comenzi. Comenzile **INDEX** și **REINDEX** lucrează cu toate articolele, indiferent de starea **SET DELETED**. Dacă este **ON** se lucrează în majoritatea comenziilor ca și cum articolele marcate pentru ștergere nu ar exista, iar **RECALL ALL** nu va găsi nici un articol



REPLACE <field1> WITH <expr1> [ADDITIVE] [, <field2> WITH <expr2> [ADDITIVE]]..[<scope>] [FOR <expL1>] [WHILE <expL2>]

REPLACE schimbă conținutul câmpurilor specificate din tabela activă, implicit doar articolul curent, cu condiția ca pointerul de articol să nu fie pe **EOF**.

ADDITIVE se referă doar la câmpurile memo și înseamnă adăugarea la vechiul conținut, fără ca acesta să se piardă. Dacă se dorește înlocuirea în câmpul care este cheie primară nu trebuie folosite opțiunile **SCOPE/FOR WHILE**.



SORT TO <file> ON <field1>[/A|/D]/[C][, <field2>[/A|/D]/[C]...][ASCENDING|DESCENDING][<scope>][FOR <expL1>][WHILE <expL2>][FIELDS <field list>|FIELDS LIKE <skel>|FIELDS EXCEPT <skel>][NOOPTIMIZE]

SORT creează o nouă tabelă, în care articolele din tabela activă sunt sortate în ordinea câmpurilor specificate.

- nu se pot face sortări după câmpuri logice sau memo;
- implicit, sortările se fac în ordine crescătoare, **A** indică ordine crescătoare, **D** descrescătoare;
- **C** impune să nu se diferențieze literele mari de cele mici și se poate combina cu **A** și **D**;
- se poate indica tipul de sortare pentru fiecare câmp în parte;
- fișierul rezultat trebuie să fie diferit ce cel de origine, și să nu fie deschis;
- dacă există deja un fișier cu numele indicat se va scrie peste acesta.



2.7 Exerciții

1. Afisați data curentă în diferite formate (prezentate în tabelul din carte). Aflați câte zile sunt până în anul 2010.
2. Aflați în ce zi cade 1 ianuarie a anului viitor și a câte zi din săptămână este.
3. Utilizați operatorii (concatenare și relaționali) definiți pentru sirurile de caractere.
4. Testați funcțiile: SUBSTR(), LEFT(), ALLTRIM(), LEN(), LOWER(), CHR().
5. Testați funcțiile de conversie între diferite tipuri de date.
6. Creați tabelul AGENDA cu structura următoare, folosind comanda CREATE TABLE .

AGENDA			
Nr.Crt.	Denumire câmp	Tip câmp	Lungime câmp
1	Cod_ID	Num	4
2	Nume	Char	25
3	Prenume	Char	25
4	Adresa	Memo	-
5	Oras_Rez	Char	15
6	Tara_Rez	Char	15
7	Data_nast	Date	-
8	Meserie	Char	15
9	Loc_Munca	Memo	-
10	Tel_a	Num	15
11	Tel_s	Num	15
12	Tel_m	Num	15
13	Fax	Num	15
14	Email	Char	25

Folosiți comanda COPY STRUCTURE EXTENDED TO și creați fișierul STRU_AGD. Consultați conținutul noului fișier. Stergeți logic și apoi fizic articolele cu numerele 4,8,9,11,12,13,14. Construiți structura unei baze de date noi, AGENDA1, folosind comanda : CREATE Agenda1

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

FROM Stru_Agd.

7. Deschideți simultan două tabele în două zone de lucru diferite. Închideți toate tabelele și ieșiți din Fox.
8. Creați tabelele de mai jos, cu indecșii specifice.
Autor.dbf (CodAutor C3 Index primar, Autor C15 Index normal),
CartiAutor.dbf (CodAutor C3 Index normal, CodCarte C3 Index normal)
Carti.dbf (CodCarte C3 Index primar, Denumire C50, AnAparitie C4, Pret N8,
CodEditura (C3 Index normal)
9. Creați o bază de date numită Bibliotecadbc care să conțină tabelele anterior create la care adăugați următoarele tabele:
Editura.dbf (CodEditura C3 Index primar, Editura C30 Index normal)
Domeniu.dbf (CodDomeniu C3 Index primar, Domeniu C30 Index normal)
10. Activăți pe rând fiecare index și vizualizați datele. Ce schimbări apar? Stergeți unul dintre indecșii asociați tabelei. Vizualizați datele! Care este efectul?

Capitolul 3 Indexarea și relaționarea tabelelor, integritatea referențială.

3.1 Indexarea tabelelor



Indexarea reprezintă operația de ordonare logică a datelor dintr-o tabelă după diferite criterii, operație ce nu afectează ordinea fizică a datelor din tabelă, ci doar modul în care acestea sunt văzute.

Ordonarea unei tabele presupune stabilirea criteriilor în care aceasta va fi parcursă.

Cheia de ordonare sau Criteriul de ordonare poate fi un câmp al tabelei sau o expresie rezultată din combinarea mai multor câmpuri.

Pentru stabilirea ordinii înregistrărilor tabeliei se evaluează cheia de ordonare pentru fiecare înregistrare în parte. Valorile obținute se compară între ele, iar ordinea înregistrărilor în tabelă se stabilește în funcție de rezultatul obținut.

Prin indexare se construiește un fișier special, numit fișier de index, ce va fi folosit în scopul regăsirii datelor în ordinea stabilită de cheia de ordonare. Un index poate fi considerat ca un filtru ce poate fi aplicat unei tabele pentru ca datele respective să poată fi privite într-o anumită ordine.

Fișierele index ce pot fi asociate unei tabele pot fi:

- fișiere index simple (extensia IDX) conțin o singura cheie de indexare;
- fișiere index simple compacte (extensia IDX)- care utilizează tehnologia de compresie dinamică de la indecsii compuși (vezi mai jos !!) ;
- fișiere index compuse (extensia CDX)- care memorează chei de indexare (etichete, tag-uri) într-un singur fișier index, la un moment dat fiind activă una singură.

O tabelă poate fi ordonată după mai multe chei, caz în care se creează mai multe intrări de indecsii pentru aceeași tabelă în același fișier de index.

La deschiderea unei tabele indexate este necesară doar deschiderea fișierului de index asociat, care conține toate informațiile despre toți indecsii. Utilizarea indecsilor se poate face prin specificarea indexului care să dea

Facultatea de Automatică și Calculatoare Iași ***Baze de date – lucrări practice***

ordinea dorită. La un moment dat doar unul dintre indecsi poate fi activ și acest index se numește index activ.

Toți indecsi asociați unei tabele sunt memorati în același fișier, care are același nume ca și tabela dar cu extensia “cdx”. Fiecare index dintr-un fișier de index î se atribuie un nume (eticheta index), nume prin care se face referirea în comenziile de prelucrare.

Indecsi pot fi de mai multe tipuri, și anume :

- **normali (Regular)** – care construiesc pentru fiecare înregistrare din tabela câte o înregistrare în fișierul index respectiv, indiferent de duplicarea sau multiplicarea valorii cheii de indexare. Într-o astfel de tabelă sunt accesibile toate înregistrările, indiferent de valoarea cheii de indexare;
- **unici (Unique)** – care permit doar o valoare unică cheii de indexare. Dacă două sau mai multe înregistrări ale tabelei au aceleasi valori ale cheii de indexare, numai prima dintre acestea va fi disponibilă, restul neapărând în tabelă (deși fizic există) ;
- **candidat** – este un index asemănător cu cel unic, dar interzice încărcarea de înregistrări ce dublează valoarea cheii de indexare, caz în care se generează un mesaj de eroare;
- **primar (cheie primara)** – în cadrul unei tabele pot exista mai multe câmpuri (sau mai multe criterii) care să asigure identificarea unică a înregistrărilor. Dintre acestea se alege unul care să fie folosit drept cheie primară a tabelei atunci când se creează o relație între tabela respectivă și o altă tabela din baza de date.

Crearea indecsilor

În fereastra pentru crearea tabelelor, în pagina Fields există un fanion de indexare care, prin poziționarea în modul “săgeată în sus” sau “săgeată în jos”, determină crearea unei etichete de index în fișierul de index pentru câmpul respectiv, cu același nume ca și câmpul, criteriul de ordonare fiind câmpul respectiv. Ordinea crescătoare sau descrescătoare este dată de tipul săgeții.

Comanda de indexare este:



INDEX ON <expr> TO <idx file>|TAG <tag name>[OF <cdx file>][FOR <expL>][COMPACT] [ASCENDING|DESCENDING] [UNIQUE] [ADDITIVE]

<expr> - cheia de indexare care conține câmpuri ale tabelei dar nu și câmpuri memo.

Clauza **FOR** permite limitarea înregistrărilor ce vor putea fi accesate prin fișierul index la cele care îndeplinesc condiția <expL> .

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Clauza **COMPACT** permite crearea fișierelor index simple compacte. Fișierele index compuse sunt întotdeauna compacte, deci clauza ar fi redundantă.

Clauzele **ASCENDING** (implicită) și **DESCENDING** stabilesc modul de ordonare a informațiilor: crescător /descrescător pentru indecsi compuși.

Clauza **UNIQUE** permite, în cazul în care sunt mai multe înregistrări cu aceeași cheie de indexare, accesarea numai a primei înregistrări din tabelă, la celelalte neavând acces.

Clauza **ADDITIVE** (nu are sens la indecsi compuși structurali) permite ca la crearea unui nou index, cel activ să nu fie închis, ceea ce se întâmplă automat în absența clauzei.

Comanda se folosește astfel pentru crearea celor 2 tipuri de index:

- index simplu:

```
INDEX ON <expr> TO <idx file> [FOR <expL>][COMPACT][UNIQUE]  
[ADDITIVE]
```

- index compus:

```
INDEX ON <expr> TAG <tag name> [OF <cdx file>] [ASCENDING |  
DESCENDING] [UNIQUE] [FOR <expL>]
```

Comanda de deschidere a unei tabele împreună cu fișierele INDEX asociate va arăta astfel:



```
USE [<file>|?][INDEX <index file list>|?][ORDER [<expN>]  
<file.idx>|[TAG]<tag name>[OF <file.cdx>][ASCENDING|  
DESCENDING]]]]
```

În urma acestei comenzi se va deschide tabela <file> împreună cu lista de fișiere index (separate prin virgulă) care sunt deja create cu comanda **INDEX**. Extensia fișierelor index nu trebuie precizată, decât dacă sunt 2 fișiere cu același nume și extensie diferită.

Clauza **ORDER** permite stabilirea fișierului indexul simplu sau eticheta dintr-un fișier index compus ce va deveni activă. <expN> este numărul de ordine al fișierului index simplu sau al etichetei din fișierul index compus. Sunt numărate mai întâi fișierele index simple din listă, apoi etichetele aparținând indexului compus structural și, în ultimul rând, etichetele aparținând altor indecsi compuși nestructurali. Dacă <expN>=0, înregistrările vor fi accesate în ordinea lor fizică (ca și cum nu ar exista un index).

Exemplu: tabela AGENDA care are mai mulți indecsi asociați:



USE agenda INDEX agenda ORDER TAG telefon

- eticheta TELEFON din indexul compus structural AGENDA.CDX- va fi indexul activ

USE agenda INDEX agenda_d ORDER TAG data_nast

- eticheta DATA_NAST din indexul compus AGENDA_D.CDX- va fi indexul activ

Dacă sunt deschise mai multe fișiere index simple sau compuse, pentru a schimba indexul (eticheta) activ(ă) se folosește comanda:



**SET ORDER TO [<expN1>|<file.idx>|[TAG]<tag name>[OF <cdx file>][IN <expN2>|<expC>][ASCENDING|DESCENDING]]
CDX(<expN1>[,<expN2>|<expC>])**

Actualizare indecși

Actualizarea indecșilor asociați unei tabele se face folosind comanda :
REINDEX

Sunt multe situații în care un index poate fi desincronizat (căderi de tensiune, probleme de alocare clusteri dubli sau inexistenti) caz în care acesta trebuie refăcut. Cei mai puțini expuși unor astfel de probleme sunt indecșii compuși strucurali, care sunt deschiși odată cu tabela asociată. și aceștia pot suferi fragmentari semnificative, în urma adăugării sau ștergerii de articole. De asemenea ștergerea unei etichete dintr-un index compus nu eliberează spațiul pe disc. Din acest motiv este recomandată utilizarea ritmică a comenzi **REINDEX**.

Ștergere indecși



**DELETE TAG <tag name1> [OF <file1.cdx>] [,<tag name2> [OF <file2.CDX>]]... sau
DELETE TAG ALL [OF <cdx file>]**

Comanda permite ștergerea etichetelor dintr-un index compus fie strucural, fie specificat de clauza **OF**. Clauza **ALL** permite ștergerea tuturor etichetelor, caz în care fișierul index compus specificat este șters de pe disc.

Regăsirea rapidă a datelor

În afara de afișarea datelor într-o anumită ordine, indecșii prezintă și avantajul major de regăsire rapidă a datelor. Căutarea rapidă se face cu comanda:

SEEK <expr>

Comanda este urmată de o expresie ce reprezintă valoarea căutată printre valorile cheii de indexare active. Dacă este găsită o asemenea înregistrare, indicatorul de înregistrări se va poziționa pe acesta, funcția **FOUND()** va returna valoarea adevărat, iar funcția **EOF()** va returna fals. În caz contrar, indicatorul de înregistrări se poziționează după ultima înregistrare **FOUND()** returnează fals iar **EOF()** returnează True.

3.2 Relationarea tabelelor

O bază de date relațională reprezintă o bază de date în care tabelele (fișierele de date -.DBF) sunt organizate și accesate prin intermediul relațiilor. O relație este o legătură între fișiere de date, legătură care permite accesarea informațiilor din mai multe fișiere (nu numai cea din zona curentă). Legătura reprezintă condiția de asociere între fișierele de date, și este materializată printr-un câmp comun fișierelor de date. Modelul relațional al bazelor de date implică organizarea datelor în tabele legate între ele prin relații. Scopul stabilirii relațiilor este acela de coordonare, după diferite criterii, a datelor din tabelele aflate în legătură.

Relațiile dintre tabele se pot clasifica în următoarele patru clase:

- **Una – la – una** – caz în care fiecare înregistrare din tabela părinte este pusă în corespondență cu o înregistrare din tabela copil;
- **Una – la – mai multe** – unei înregistrări din tabela părinte îi corespund mai multe înregistrări din tabela copil;
- **Mai multe- la- una-** unei înregistrări din tabela copil îi corespund mai multe înregistrări din tabela părinte;
- **Mai multe- la- mai multe-** unei înregistrări din tabela părinte îi corespund mai multe înregistrări din tabela copil și invers, unei înregistrări din tabela copil îi corespund mai multe înregistrări din tabela părinte.

În Visual Fox Pro sunt implementate doar primele trei tipuri de relații, cea de a patra putând fi redusă la două relații și anume: una de tip **mai multe – la – una** și alta de tip **una – la – mai multe** prin intermediul unei tabele suplimentare intercalată între cele două. Din punct de vedere al momentului definirii relațiilor dintre tabele, există :

Facultatea de Automatică și Calculatoare Iași ***Baze de date – lucrări practice***

- **Relații temporare sau dinamice**, care se creează prin comenzi în timpul rulării programelor și sunt disponibile numai în timpul rulării acestora;
- **Relații permanente**, ce se creează în faza de proiectare a bazei de date și sunt deschise automat de către sistem odată cu deschiderea bazei de date;

Pentru crearea relațiilor permanente între două tabele este necesar ca:

- Tabela părinte să fi indexată cu un index candidat sau primar;
- Tabela copil să fie indexată cu orice fel de index ;
- Indecșii corespunzători trebuie să existe anterior creării relației.

Observații : Tabela părinte indexată cu un index candidat sau primar determină stabilirea numai a unor *relații permanente* de tip una – la una sau una – la – mai multe, deoarece în tabela părinte, după indexul respectiv, nu pot exista mai multe înregistrări cu aceeași valoare a cheii primare.



Presupunând ca baza de date este deschisă și tabelele sunt corespunzător indexate, stabilirea unei relații permanente se face printr-un click pe indexul primar sau candidat în tabela părinte după care se trage mouse-ul peste indexul din tabela copil.

De exemplu, avem tabelele AG(enti) și TR(anzacții). Agentii care au intermediat tranzacții se regăsesc în fișierul TR, prin intermediul câmpului NR_AGENT și nu prin intermediul câmpurilor NUME și PRENUME. Deci se poate stabili o relație între cele două tabele prin intermediul câmpului NR_AGENT, care aparține celor 2 fișiere. Acest lucru se poate efectua prin comanda :



***SET RELATION TO [<expr1> INTO <expN1>|<expC1>
[,<expr2> INTO <expN2>|<expC2>...]....[ADDITIVE]]***

În acest exemplu există o **tabelă părinte** (cea activă în momentul comenzi și anume Agentii) și una sau mai multe **tabele copii** (Tranzacții) specificate prin număr <expN_i> sau alias <expC_i>, toate aceste tabele fiind deschise în zone de lucru diferite.

Cheia relației este stabilită prin <expr_i> care corespunde unui câmp (sau la mai multe câmpuri) din tabela părinte. **Cheia relației este și cheia de indexare a tabelei copil**.

Stabilirea unei relații între 2 tabele presupune că mutarea pointer-lui de articol în tabela părinte determină și mutarea pointer-ului în tabela copil. Bineînteles că inițial se evaluatează <expr_i> și dacă sunt găsite înregistrările corespunzătoare în tabela copil, pointer-ul va fi mutat în consecință, altfel va ajunge la sfârșitul tabelei și deci EOF() pentru zona de lucru respectiva va fi .T.

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

Clauza ADDITIVE face ca relațiile deja existente pentru TABELA activă să nu fie șterse.

Această comandă face ca fiecare înregistrare din tabela părinte să-i corespundă o înregistrare în tabela copil, unde saltul se face automat. Se pune întrebarea, ce se întâmplă când unei înregistrări părinte îi corespund mai multe articole în tabela copil (care nu are activ un index unic). În aceasta situație comanda determină numai găsirea primei înregistrări, deci există o relație “ONE TO ONE” ⇔ “UNA LA UNA”.

Comanda care permite găsirea tuturor înregistrărilor din tabela copil ce corespund unei singure înregistrări din tabela părinte, adică o relație tip “ONE TO MANY” ⇔ “UNA LA MAI MULTE” este:

SET SKIP TO [<alias1> [,alias2>]...]

Pentru a folosi, relațiile trebuie să fie deja create cu o comandă

SET RELATION TO...

Modul de lucru al comenzi este următorul: dacă se mută pointer-ul în tabela părinte în urma unor comenzi **GOTO**, **LOCATE** sau **SEEK**, pointerul se mută pe articolul corespunzător și în tabela copil. Următoarele salturi în tabela părinte cu **SKIP** sau cu o buclă **SCAN.. ENDSCAN** nu vor determina mutarea pointer-ului, în schimb în tabela copil vom fi poziționați pe următoarele înregistrări ce au aceeași cheie cu înregistrarea din părinte.

Pentru ștergerea relațiilor dintre tabele se va folosi comanda:



SET RELATION OFF INTO <expN>|<expC>

care șterge relația dintre tabela curentă și tabela copil specificată prin număr -<expN> sau alias -<expC>.

SET RELATION TO folosit fără parametri determină înălțatularea tuturor relațiilor existente pentru tabela curentă.

Exemplu: se indexează inițial cele 2 tabele



```
CLEAR  
CLOSE DATABASES  
USE AG ORDER tag AG_I în 0 && Tabelul părinte  
BROW  
USE TR ORDER tag AG_I în 0 && Tabelul copil  
SELE TR  
BROW  
SELECT AG    && Selectare tabel părinte
```

Facultatea de Automatică și Calculatoare Iași ***Baze de date – lucrări practice***

```
SET RELATION TO nr_agent INTO TR    && Stabilirea relației  
SET SKIP TO TR && Tipul relației este 'One-to-many'  
WAIT WINDOW 'Afisarea TRANZACTIILOR efectuate de fiecare  
AGENT' BROWSE FIELDS AG.NR_AGENT :H='Nr.Id.',AG.NUME  
:H='Nume', ; AG.PRENUME :H='Prenume', TR.NR_TRANZ
```

3.3 Integritatea referențială

Integritatea referențială reprezintă un ansamblu de reguli impuse tabelelor între care s-au stabilit relații. Aceste reguli sunt necesare deoarece modificarea datelor dintr-o tabelă poate afecta relația cu tabela legată. Evenimentele ce duc la modificări ale cheii și reprezintă evenimente tratate prin integritate referențială sunt:

- Adăugarea de noi înregistrări;
- Ștergerea unor înregistrări;
- Modificarea datelor unei înregistrări care afectează relația.

În cazul în care se **adăugă** o nouă înregistrare în tabela copil a unei relații se pot specifica următoarele opțiuni:

- ✓ **Ignorare** – se pot introduce valori, indiferent dacă există sau nu înregistrări corespunzătoare în tabela părinte;
- ✓ **Restrictionare** – se generează un mesaj de eroare atunci când se încearcă adăugarea unei înregistrări în tabela copil care nu are corespondent în tabela părinte.

În cazul în care se **șterg** înregistrări din tabela părinte a unei relații se pot specifica următoarele opțiuni:

- ✓ **Ignorare** – se permite ștergerea, chiar dacă în tabela copil există sau nu înregistrări legate de cea ștearsă;
- ✓ **Restrictionare** – se generează un mesaj de eroare atunci există corespondent în tabela copil;
- ✓ **Ștergere în cascadă** – se șterg automat atât înregistrările din tabela copil cât și cele din tabela părinte.

În cazul în care se face o **modificare** care afectează relația în tabela părinte se pot specifica următoarele opțiuni:

- ✓ **Ignorare** – se permit modificările respective, chiar dacă în tabela copil există sau nu înregistrări legate de cea modificată;
- ✓ **Restrictionare** – se generează un mesaj de eroare atunci există corespondent în tabela copil;
- ✓ **Modificare în cascadă** – se modifică automat atât înregistrările din tabela copil cât și cele din tabela părinte conform noii valori a cheii relației.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Opțiunile legate de integritatea referențială se setează prin accesarea ferestrei **Constructorului de baze de date** în care se execută dublu click pe relația respectivă (linia frântă care leagă tabelele). Se deschide fereastra **Edit Relationship** în care se apăsa pe butonul **Referențial Integrity** care determină deschiderea unei alte ferestre de unde se pot alege opțiunile dorite.

Cele trei pagini ale ferestrei controlează fiecare câte un tip de eveniment astfel :

- Rules for Updating** – pagina cu opțiuni referitoarea la modificarea datelor din tabelele legate;
- Rules for Deleting** – pagina cu opțiuni referitoarea la ștergerea datelor din tabelele legate;
- Rules for Inserting** – pagina cu opțiuni referitoarea la adăugarea datelor din tabelele legate.

3.4 Funcții. Variabile de memorie. Macrosubstituția

Variabile de memorie reprezintă zona de memorie căreia î se atribuie un nume și care poate stoca o valoare de un anumit tip. Deci elementele unei variabile sunt:

- numele – folosit pentru identificare și atribuit de programator sau predefinit de proiectanții FoxPro. Numele variabilelor este format din maxim 8 caractere de tip literă, cifră sau caracterul '_', cu primul caracter diferit de cifră;
- conținutul sau valoarea variabilei – reprezintă data ce este memorată în zona de memorie a variabilei;
- tipul variabilei – tipul datei ce se poate memora în zona de memorie respectivă.

Alocarea memoriei pentru variabile se face asemănător ca în limbajul BASIC, adică, pentru variabile simple, în momentul primei atribuirii a valorii se aloca spațiu pentru variabilă. Variabilele tablou (masivele) trebuie declarate în prealabil și pot avea maxim 2 dimensiuni. Numărul de elemente pe o dimensiune se poate preciza și prin expresii. În plus, în limbajul FoxPro caracteristica de tip de date este asociată elementului tablou și nu tabloului, de unde rezultă ca elementele același tablou pot fi tipuri diferite.

Crearea unei variabile sau modificarea valorii acesteia se face prin operatorul de atribuire:



<var> = <e> sau prin comanda
STORE <e> TO <var>

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

- se evaluează expresia **<e>**, obținându-se o valoare de un anumit tip;
- se caută în memorie variabila cu numele **<var>** și, dacă se găsește, se înlocuiește vechiul conținut al acesteia cu valoarea expresiei;
- dacă nu se găsește variabila respectivă, FoxPro creează una nouă cu numele **<var>**, în care depune valoarea expresiei;
- tipul variabilei este dat de tipul valorii expresiei, indiferent de tipul anterior al variabilei, în cazul în care aceasta există și înainte de execuția comenzi;
- de obicei pentru variabilele de memorie se folosește simbolul **M**

O tehnică specială de lucru cu variabile o reprezintă **macrosubstituția**, prin care conținutul unei variabile de tip sir de caractere este tratat ca nume al altei variabile sau al altui element FoxPro (câmp al unei baze de date, denumire fișier). Macrosubstituția funcționează ca și cum în locul variabilei respective ar fi pus sirul de caractere conținut de aceasta, fără apostrofurile delimitatoare. Operatorul de macrosubstituție este **&**.

Exemplu :



```
test = "* .dbf"
DIR &test
ACCEPT 'Cu ce fișier doriți sa lucrați ? ' TO test
USE &test
x="italian"
SET DATE TO &x
EDIT
```

Variabilele de memorie pot fi locale sau globale și se definesc cu comenziile:

PUBLIC <lista variabile> - variabilele din listă se declară publice

PRIVATE <lista variabile> - variabilele din listă se declară ca fiind locale

Ștergerea variabilelor de memorie se face cu comanda :



RELEASE <memvar list>
RELEASE ALL [LIKE /EXCEPT <skeleton>]

Lista conține numele variabilelor la care se renunță, separate prin virgulă.

RELE ALL șterge toate variabilele de memorie, mai puțin cele sistem. Într-un program, **RELE ALL** șterge doar variabilele locale, create în programul curent sau în cele de nivel inferior.

Afișarea de informații despre conținutul memoriei FOX se face folosind comanda :



LIST / DISPLAY MEMORY [TO PRINTER / TO FILE <file name>]

care determină afișarea variabilelor de memorie și utilizator existente, tipul și valoarea lor.

Diferența între **LIST** și **DISPLAY** este că la aceasta din urmă se așteaptă apăsarea unei taste după umplerea unui ecran cu informații. Se poate face redirectarea informațiilor către imprimantă sau către un fișier.

Instrucțiuni de tip SCATTER/GATHER



SCATTER [FIELDS lista câmpuri] [MEMO]TO <masiv> | TO <masiv> BLANK | MEMVAR | MEMVAR BLANK

Comanda permite copierea datelor din articolul curent într-un masiv sau un set de variabile de memorie (pe care le creează dacă nu sunt definite deja). Clauza **FIELDS** permite copierea numai pentru datele din câmpurile specificate în listă. Se pot prelua date și din câmpuri tip memo dacă este prevăzuta clauza **MEMO**, altfel aceste câmpuri sunt ignorate. Clauza **BLANK** permite crearea unui masiv sau variabile de memorie cu elemente "goale" (sesizabile cu funcția `EMPTY()`). Clauza **MEMVAR** permite crearea unui set de variabile de memorie care au același nume și tip cu câmpurile bazei de date care însă sunt precedate de calificatorul "*m*".

GATHER FROM <masiv> | MEMVAR FIELDS <lista câmpuri>][MEMO]

Comanda permite scrierea valorilor dintr-un masiv sau set de variabile de memorie în câmpurile înregistrării curente din tabela activă. Clauzele **FIELDS** și **MEMO** au aceeași semnificație ca și la **SCATTER**. De asemenea numele variabilelor de memorie respectă convenția de la **SCATTER**.

Utilizatorii pot să-și definească propriile funcții (UDF = User Define Function) folosind comanda:



FUNCTION <function name>

Modalitatea de transmitere a parametrilor este aceeași cu cea prezentată la proceduri.

Scrierea codului se încheie obligatoriu cu **RETURN <exp>**. După execuția funcției, FOX returnează programului apelant valoarea calculată de funcție.

Exemplu: creați un fișier TEST.prg cu următorul conținut:



```
PROCEDURE TEST
PARAMETERS X,Y, suma,produs
suma=adunare(x,y)
produs=înmulțire(x,y)
RETURN
FUNCTION ADUNARE
PARAMETERS x, y
RETURN x+y
FUNCTION înmulțire
PARAMETERS x,y
RETURN x*y
```

și testați-l cu parametrii s=0 p=0

```
DO TEST WITH 3, 4, s, p
? s
? p
? adunare (5, 6)
SET PROCEDURE TO TEST
? adunare (3, 4)
? înmulțire (5, 6)
SET PROCEDURE TO TEST
? înmulțire (3, 3)
```

3.5 Crearea și modificarea programelor



MODIFY COMMAND /FILE <filename> [WINDOW <window name>]

Comanda apelează editorul de texte din FOX. În varianta **COMMAND** se caută implicit extensia.PRG. Dacă fișierul indicat nu există, acesta se creează. Se scriu liniile de comandă iar la tastarea **CTRL/ END** se încheie editarea, se salvează fișierul pe disc și se revine la prompter. Meniul oferă comenzi similare.

Etapele elaborării și rulării unui program sunt următoarele:

- se concepe algoritmul de rezolvare a problemei propuse;
- se transpune algoritmul într-un sir de instrucții ce reprezintă chiar programul;
- se introduce programul într-un fișier;
- se compilează programul;

- se execută programul.

Exemplu : să considerăm următorul program, numit “PRIMUL”



MODI COMM PRIMUL
Clear
@ 14,30 say 'Primul program scris in Visual Fox Pro '

Programul va șterge ecranul și va afișa în mijlocul său mesajul ‘Primul program scris in Visual Fox Pro’. Pentru compilare se dă comanda : **COMPILE PRIMUL** (compilarea se face de obicei automat, deci aceasta comandă nu este necesară decât pentru cunoașterea modului de funcționare al Fox) iar execuția (rularea) se efectuează cu comanda: **DO PRIMUL**.

Instrucțiunile din cadrul unui program pot fi scrise cu litere mari sau mici. O instrucțiune se poate scrie și pe mai multe rânduri, caz în care linia curentă se încheie cu caracterul ‘;’. În cadrul unei linii se pot introduce și comentarii specificate de caracterele: ‘**’, ‘&&’ sau comanda : **NOTE / * <text> [...]**

3.6 Programare structurată. Proceduri.

Apelarea, transmiterea parametrilor, revenirea, compilarea

Un program reprezintă o succesiune de instrucțiuni realizată în conformitate cu regulile limbajului de programare folosit, care rezolvă o anumită problemă. Un program FOX este scris pe disc într-un fișier. Când se dorește execuția lui, fișierul este compilat și apoi rulat.
Execuția unui program se realizează cu comanda **DO** având sintaxa :

DO <nume fișier> [WITH <lista parametri>] [IN <fișier>]

Aceasta comandă execută instrucțiunile conținute în fișierul <fișier>. Dacă <fișier> nu are prevăzută extensia, atunci se vor căuta pe disc următoarele fișiere, în această ordine:

- <fișier>.EXE - program executabil
- <fișier>.APP - aplicație
- <fișier>.FXP - forma compilată a programului
- <fișier>.PRG - programul sursă

Modul de execuție a unui program poate depinde de parametrii externi care se transmit programului în momentul execuției. De exemplu, presupunem un program de sortare a datelor dintr-un fișier care necesită ca parametri

Facultatea de Automatică și Calculatoare Iași ***Baze de date – lucrări practice***

externi numele fișierului și ordinea sortării. La execuția comenzi **DO** parametrii sunt transmiși către program prin lista de parametri a clauzei **WITH**. Această listă poate conține expresii, constante, câmpuri ale unui tabel de date, separate prin virgulă.

Execuția unui program se va opri în una din următoarele situații:

- la execuția unei comenzi **RETURN**, **CANCEL**, **QUIT** când se întâlnește sfârșitul fișierului;
- când se întâlnește o alta comandă **DO** ;
- prin apăsarea tastei **ESC** dacă **SET ESCAPE** este **ON**.

Execuția unui program din interiorul altui program folosind comanda **DO** reprezintă un pas important în structurarea aplicațiilor de dimensiuni mari. Pe măsură ce dimensiunea unei aplicații crește, testarea și depanarea acesta devine din ce în ce mai dificilă datorită numărului mare de variabile și instrucțiuni folosite.

Utilizatorul poate defini anumite proceduri și funcții ce pot fi executate de mai multe ori în cadrul unui program, apelându-le din diverse puncte de program, cu diverse valori ale parametrilor.

Scrierea unei proceduri începe cu :

PROCEDURE <procedure name> și se încheie obligatoriu cu **RETURN**.

Procedurile și funcțiile unui program se introduc, de regulă, după ultima instrucțiune a programului, în același fișier cu acesta. Acestea pot fi introduse și în fișiere separate care vor fi asociate programului apelant prin comanda:

SET PROCEDURE TO [<fișier>]

Această comandă asociază programului curent în execuție, fișierul <fișier> unde se vor căuta procedurile ce nu se găsesc în programul apelant.



Parametrii sunt variabile de comunicare între programe, definite de utilizator, ce realizează interfața programului cu exteriorul.

Într-o procedură se pot defini două tipuri de variabile :

- **globale (publice)**, ce pot fi accesate și modificate în orice procedură în curs de execuție **PUBLIC <lista variabile>**;
- **locale (private)**, ce nu pot fi accesate decât în procedura curentă și în cele subordonate acesteia **PRIVATE <lista variabile>**.

Exemplu:



```
SET TALK OFF
CLEAR
PRIVATE a
PUBLIC b
a=1
b=2
DO TEST
NOTE aici se cunosc variabilele a,b,d dar nu și c
? a
? b
? c
? d
WAIT ''
RETURN
PROCEDURE TEST
PRIVATE c
PUBLIC d
c=3
d=4
NOTE aici se cunosc toate variabilele a,b,c,d
? a
? b
? c
? d
WAIT ''
RETURN
```

RETURN [<expresion> / TO MASTER /TO <procedure name>]

Instrucțiunea returnează controlul programului apelant sau la prompter.

TO MASTER permite revenirea în programul apelant de cel mai înalt nivel.

TO <procedure name> permite revenirea în procedura activă indicată la sfârșitul unei funcții și întoarcerea unui rezultat.

3.7 Comenzi de intrare/ieșire



```
? /?? [<exp1> ][PICTURE <expC1>][FUNCTION <expC2>]
[AT <expN2> ]
```

Comanda afișează valoarea pentru una sau mai multe expresii separate prin virgulă. ? produce afișarea pe linia următoare în timp ce ?? afișează pe linia curentă.

Dacă **SET PRINT** este **ON** ieșirea comenzi este dirijată către imprimantă.

PICURE și **FUNCTION** formatează afișarea ieșirii (vezi @....**SAY....GET**) iar **AT** stabilește coloana în care sa se facă afișarea.



ACCEPT <expC> TO <memvar>

Afișează (eventual) mesajul **<expC>**, care poate fi o variabilă de memorie de tip caracter sau un sir de caractere încadrat între ghilimele. Răspunsul utilizatorului, terminat cu **ENTER**, este stocat în variabila de memorie indicată (dacă este cazul, aceasta este creată). Apăsarea tastei **ESC** (dacă se lucrează cu **SET ESCAPE ON**) duce la terminarea execuției comenzi.

WAIT [<expC>] [TO <memvar>]

Afișează un mesaj care implicit este “Press any key to continue....” și care așteaptă apăsarea unei taste (optional memorate în variabila de memorie specificată).



Comanda **@ SAY... GET** creează formate particulare pentru intrare/ ieșire.

@ <row, column> SAY <expr>[FUNCTION <expC1>] [PICTURE <expC2>] [SIZE <expN1>, <expN2>][FONT <expC3> [, <expN3>]] [STYLE <expC4>] [COLOR SCHEME <expN4>] COLOR <color pair list>]

@ <row, column> GET <memvar> | <field>[FUNCTION <expC1>] [PICTURE <expC2>] [FONT <expC3> [, <expN1>]] [STYLE <expC4>] [DEFAULT <expr1>] [ENABLE | DISABLE] [MESSAGE <expC5>] [[OPEN] WINDOW <window name>] [RANGE [<expr2>] [, <expr3>]] [SIZE <expN2>, <expN3>] [VALID <expL1> | <expN4>] [ERROR <expC6>] [WHEN <expL2>] [COLOR SCHEME <expN5>] COLOR <color pair list>]

- **SAY** afișează informații iar **GET** permite introducerea sau editarea valorii datelor;
- **row** și **column** sunt expresii numerice (se poate folosi și ‘\$’ pentru poziția curentă) ;
- coordonatele sunt totdeauna relative la colțul din stânga sus al ecranului sau al ferestrei active (0,0) ;
- **GET** permite afișarea și actualizarea datelor conținute în câmpuri sau atribuite unor variabile de memorie temporare ;
- comanda **READ** activează toate **GET** anterioare ;

- dacă **READ** este comun pentru mai multe **GET**, se trece la următoarea citire prin **ENTER** sau săgeata în jos;
- dacă **PICTURE** indică mai multe caractere decât cele specificate la crearea variabilei de memorie sau a câmpului, se acceptă la citire dar nu se memorează toate;
- ștergerea unor caractere se face cu **BackSpace** sau **Del**;
- folosind clauza **COLOR** din comandă se pot stabili culorile de afișare;
- clauza **WHEN** indică o condiție ce va fi evaluată când încercam să poziționam cursorul pe un câmp **GET**; dacă este falsă, se trece la următorul câmp;
- **DEFAULT** furnizează o valoare implicită pentru **GET**; expresia trebuie să corespundă tipului de dată din **GET**, ce nu este evaluată decât la adăugarea înregistrărilor în tabelă;
- **RANGE** stabilește intervalul închis de valori permise. Se poate indica numai limita inferioară sau numai cea superioară;
- dacă se introduce o valoare incorectă, se afișează mesajul corespunzător și se așteaptă apăsarea tastei **Space** după care se cere o nouă valoare (până la satisfacerea restricțiilor) ;
- **VALID** stabilește condiția care trebuie să fie îndeplinită înainte ca datele să fie acceptate de **GET** (se poate folosi o funcție utilizator care să returneze o valoare logică ; nu se face nici un control dacă se apasă direct **ENTER**) ;
- **ERROR** indică un mesaj personal care să fie afișat dacă nu s-a satisfăcut validarea;
- **MESSAGE** este o expresie caracter afișată la execuția **READ** când cursorul este plasat în câmpul **GET** asociat acestui mesaj;
- **WINDOW** permite deschiderea unei ferestre pentru editarea câmpurilor **MEMO**; dacă se indică și **OPEN** fereastra se deschide automat;
- **PICTURE** se folosește pentru a limita tipul datelor ce pot fi încărcate (de exemplu doar cifre) și pentru a formața afișarea. Se pot folosi măști sau funcții de formatare pentru introducerea datelor, și anume:

Măști

- ! convertește litere mici în litere mari;
- # permite doar cifre, spații și semn;
- \$ afișează sirul de caractere corespunzător lui SET CURRENCY în locul zerourilor nesemnificative;
- * afișează asteriscuri în locul zerourilor nesemnificative;

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

- , afișează virgula dacă există cifre la stânga ei;
- . indică poziția zecimală;
- A autorizează doar litere;
- L autorizează doar date logice;
- N autorizează doar litere și cifre;
- X orice caracter;
- Y autorizează doar date logice;

Masca trebuie să aibă atât de caractere câte sunt specificate în structura câmpului respectiv. Dacă se lucrează cu numere, în masă trebuie inclus și punctul zecimal și trebuie lăsat loc și pentru semn.

Funcții

- ! acceptă orice literă și convertește litere mici în litere mari;
- # permite doar cifre, spații și semne;
- \$ afișează numerele în format monetar;
- (plasează numerele negative în paranteze;
- A autorizează doar litere;
- B aliniază la stânga textul din interiorul câmpului;
- D afișează data în formatul curent;
- I centrează textul în interiorul câmpului;
- J aliniază la dreapta textul din interiorul câmpului;
- T elibera spațiile de la începutul și de la sfârșitul câmpului;
- Z afișează spații în locul valorilor nule;

Exemplu :



```
CLEAR
SET TALK OFF
NUME=SPACE(10)
VARSTA=0
DO WHILE VARSTA <> 99
@ 6,5 SAY "NUMELE : " GET NUME PICTURE "AAAAAAA"
@ 8,5 SAY "VARSTA : " GET VARSTA PICTURE "99"
MESSAGE "pentru ieșire : vârsta este de 99 ani"
READ
@ 10,5 SAY NUME+" ARE "
@ 10, $+1 SAY VARSTA
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

```
@ 10, $+1 SAY "ANI"  
NUME=SPACE(10)  
CLEAR  
ENDDO  
WAIT 'APASATI ORICE TASTA'  
SET TALK ON  
CLEAR  
SET TALK OFF  
ZI=SPACE(8)  
DT=DATE()  
@ 1,1 SAY "DATA DE ASTAZI ESTE (USA) "  
@ 1, $+2 SAY DT PICTURE "@D"  
@ 1, $+2 SAY "SAU EUROPA"  
@ 1, $+2 SAY DT PICTURE "@E"  
@ 3,1 SAY "CE ZI ESTE ASTAZI-SPATIU SAU PRIMA LITERA  
PENTRU SELECTARE și APOI ENTER ? " GET ZI;  
FUNCTION "M LUNI, MARTI, MIERCURI, JOI, VINERI,  
SAMBATA, DUMINICA ";  
READ  
@ 5,1 SAY "ASTAZI ESTE : "+ZI+ " ! APASATI O TASTA...."  
WAIT ""  
SET TALK ON
```



READ [SAVE]

Activează toate comenziile @... **GET** executate după ultima comandă **CLEAR**, **CLEAR GETS**, **CLEAR ALL** sau **READ**.

SAVE face să nu se dezactiveze instrucțiunile **GET** la următoarea execuție pentru **READ**.

CLEAR GETS

Șterge toate comenziile @... **GET** emise după ultima comandă **CLEAR**, **CLEAR ALL**, **CLEAR GETS** sau **READ** dar nu șterge variabilele de memorie sau elementele de tablou.

3.8 Controlul fluxului

 **DO CASE**
CASE <expL1> <statements>
[CASE <expL2> <statements>]
.....
[OTHERWISE <statements>]
ENDCASE

Se execută comenziile asociate primei condiții satisfăcute (dacă nu este nici una, se execută, dacă clauza **OTHERWISE** și comenziile asociate acesteia).

Exemplu:

 **SET TALK OFF**
CLEAR
DIMENSION A[4]
A[1] = "PRIMAVARA"
A[2] = "VARA"
A[3] = "TOAMNA"
A[4] = "IARNA"
ANOTIMP = A[1]
@ 6,26 GET ANOTIMP FROM A FUNCTION "&"
READ
@ 15,10 SAY "ACEST ANOTIMP CONTINE LUNILE:"
DO CASE
CASE ANOTIMP = "PRIMAVARA"
?? "MARTIE,APRILIE,MAI"
CASE ANOTIMP = "VARA"
?? "IUNIE,IULIE,AUGUST"
CASE ANOTIMP = "TOAMNA"
?? "SEPTEMBRIE,OCTOMBRIE,NOIEMBRIE"
OTHERWISE
?? "DECEMBRIE,IANUARIE,FEBRUARIE"
ENDCASE

 **DO WHILE <expL> <statements>**
[LOOP] [EXIT]
.....
ENDDO

Se execută comenziile atât timp cât condiția este adevărată.

- **ENDDO** și **LOOP** redau controlul programului spre comanda **DO WHILE** pentru ca acesta să revalueze condiția.
- **EXIT** transmite controlul instrucțiunii care urmează după **ENDDO** (obligatorie pentru o buclă gen **DO WHILE.T.**);
- Sunt autorizate structurile **DO WHILE** imbicate.

EXEMPLU:



```
CLEAR
SUMA=0
P=1
DO WHILE P<101
SUMA=SUMA+P
P=P+1
ENDDO
SUMA=SUMA-P
A=P-1
? "S-AU ADUNAT PRIMELE ",A,"NUMERE NATURALE"
? "OBTININDU-SE SUMA DE ",SUMA
IF <expL>
  <statements>
[ELSE]
  statements>
ENDIF
```



- poate fi executată doar din interiorul unui program, nu și la prompter;
- sunt permise IF-uri imbicate;
- **ELSE** se referă la IF-ul imediat anterior;
- comanda nu poate fi executată dacă **IF** și **ELSE** sunt scrise pe aceeași linie;

EXEMPLU:

```
SET TALK OFF
CLEAR
A=0
B=0
@ 4,10 SAY "PRIMUL NUMAR " GET A PICTURE "9999"
@ 5,10 SAY "AL DOILEA NUMAR " GET B PICTURE "9999"
READ
IF A>B
  ? "Primul Număr Este Mai Mare "
ELSE
  ? "Al Doilea Număr Este Mai Mare Sau Numerele Sunt Egale"
ENDIF
IF B<>0
  ? A,';',B,'=',A/B
ENDIF
```



IIF(<expL>, <expr1>, <expr2>)

- Returnează rezultatul primei expresii dacă este adevărată condiția, respectiv rezultatul celei de-a doua condiții pentru fals.
- **<exp1>** și **<exp2>** trebuie să fie de același tip.

- Se poate utiliza de la prompter.



EXEMPLU:

SET TALK OFF
CLEAR

A=0

B=0

```
@ 4,10 SAY "PRIMUL NUMAR " GET A PICTURE "9999"  
@ 5,10 SAY "AL DOILEA NUMAR " GET B PICTURE "9999"  
READ  
SIR1="Primul Numar Este Mai Mare "  
SIR2="Al Doilea Numar Este Mai Mare Sau Numerele Sunt Egale"  
? IIF (A>B, SIR1,SIR2)  
RETURN
```

3.9 Comenzi SQL

O interogare SQL nu este neapărat o întrebare pusă bazei de date. Ea poate fi și o comandă pentru executarea uneia dintre acțiunile următoare:

- construirea unei tabele;
- inserarea, modificarea sau să ștergerea de linii sau câmpuri;
- căutarea în câteva tabele o anumitor informații și returnarea rezultatelor într-o anumită ordine;

Interogare obișnuită în SQL poate fi scrisă cu litere mari sau mici iar cuvintele cheie sunt:

SELECT.... FROM.... WHERE

Exemplu:



Creăm un tabel numit Personal, cu structura Nume C(20), Salar N(10) și dorim afișarea tuturor salariaților a căror nume este Popescu.:

SELECT NUME, SALAR FROM PERSONAL WHERE NUME='POPESCU'

Cea mai simplă interogare este : **SELECT * FROM PERSONAL**

prin care se afișează întreg conținutul tabelei. Semnul (*) transmite comanda de a returna toate coloanele din tabelul descris în clauza **FROM**.

Dacă se dorește afișarea coloanelor în altă ordine putem specifica după clauza **SELECT** numele câmpurilor dorite separate prin virgulă și spațiu.

SELECT SALAR, NUME FROM PERSONAL

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Dacă se dorește afișarea unică a anumitor valori se folosește clauza **DISTINCT**.

SELECT DISTINCT SALAR FROM PERSONAL

Dacă se dorește afișarea doar a anumitor coloane, acestea se vor specifica în clauza **SELECT**.

SELECT SALAR FROM PERSONAL

Clauza WHERE

WHERE <condiție>

Compară valoarea condiției specificate cu datele supuse interogării și returnează doar înregistrările ce satisfac condiția.



Exemplu:

Care sunt agenții ce au salariul mai mare decât 1000 ?

SELECT NUME, NR_AGENT FROM AGENTI WHERE SAL_BAZA > 1000

Asocierea tabelelor se realizează prin potrivirea valorilor unei coloane din primul tabel cu valorile unei coloane din cel de-al doilea tabel. Numele de coloane trebuie să fie unice. Numele ce nu sunt unice trebuie prefixate cu numele tabelului din care face parte, altfel apare eroare.

Exemplu:

Care sunt agenții care au realizat vreo tranzacție ?

**SELECT DISTINCT NUME, AGENTI.NR_AGENT
FROM AGENTI, TRANZACT
WHERE AGENTI.NR_AGENT=TRANZACT.NR_AGENT**

Clauza ORDER BY

ORDER(<expC1>[,<expC2>])[ASC/DESC]

Permite ordonarea afișării liniilor rezultate după valorile din coloanele indicate în această clauză. Clauza **ORDER BY** nu poate fi folosită într-o subinterrogare.

ASC (crescător-implicit) **DESC** (descrescător)

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

Exemplu:



Ordonarea agenților crescător după nume, prenume și descrescător după salariu.

```
SELECT NUME, PRENUME, SAL_BAZA  
FROM AGENTI  
ORDER BY NUME ASC,PRENUME, SAL_BAZA DESC
```



Exemple de interogări:

Care sunt numerele de cod ale agenților care au contractat tranzacții cu dischete ?

```
SELECT NR_AGENT  
FROM TRANZACT  
WHERE COD_PRODUS IN  
(SELECT COD_PRODUS FROM MARFURI WHERE  
DENUMIRE="DISCHETE")
```

Care sunt numele agenților din Iași care lucrează cu clienți de tip regii autonome?

```
SELECT NUME, NR_AGENT, ORAS, DENUMIRE  
FROM AGENTI, CLIENTI  
WHERE ORAS="IASI" AND TIP="RA"
```

Care sunt clienții din localitățile în care nu domiciliază nici un agent comercial ?

```
SELECT DENUMIRE, NR_CLIENT, ORAS, JUDET  
FROM CLIENTI  
WHERE ORAS NOT IN (SELECT ORAS FROM AGENTI )
```

3.10 Depanarea programelor

În scrierea unui program deseori apar erori datorate fie neatenției, fie unei insuficiente cunoașteri a limbajului de programare sau a omiterii unor cazuri particulare de evoluție a programului.

Există două mari clase de erori de programare :

- erori de sintaxă ;
- erori de rulare, când programul este corect scris dar nu funcționează aşa cum dorim .

La rularea unui program, apariția unei erori de sintaxă determină întreruperea execuției și afișarea unui mesaj de eroare, indicând tipul erorii și

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

cerând utilizatorului luarea unei decizii astfel :

Cancel - Întreruperea implicită a programului și revenire în prompter;

Suspend - Suspendarea execuției programului;

Ignore - Continuarea execuției, ignorând eroarea apărută.

Detectarea erorilor de rulare presupune tehnici mai avansate cum ar fi:

- rularea pas cu pas;
- folosirea punctelor de întrerupere și vizualizarea conținutului variabilelor în paralel cu execuția programului.

Toate acestea se pot efectua cu ajutorul a două ferestre sistem numite TRACE și Watch, accesibile din meniul Tools, opțiunea Debugger din FOX.

Fereastra TRACE

Pentru a depana un program, trebuie mai întâi să-l deschidem cu OPEN din fereastra TRACE. Execuția programului poate fi făcută în mai multe variante:

Pas cu pas-optiunea OVER sau STEP

- cu OVER se execută câte o instrucțiune, dar când aceasta este un apel către o rutină, acesta este executată în întregime, într-un singur pas;
- cu STEP se execută câte o instrucțiune, dar când aceasta este un apel către o rutină, acesta este executată pas cu pas;
- în combinație cu STEP se poate folosi OUT care determină executarea continuă a restului de instrucțiuni din programul curent până la un punct de întrerupere ce va determina suspendarea programului.

Punctele de întrerupere pot fi desemnate de utilizator prin deplasarea cursorului pe linia dorită și tastarea lui SPACE, ENTER sau click pe mouse. O nouă acțiune a acestor taste, când cursorul se află pe o linie ce conține un punct de întrerupere, va determina anularea acestuia.

Stergerea tuturor punctelor de întrerupere se face cu CLEAR ALL BREAKPOINTS.

Un punct de întrerupere nu determină terminarea execuției programului ci doar suspendarea acestuia, până la o nouă comandă a utilizatorului.

Execuție continuă, la viteză maximă sau execuție continuă, cu viteza controlată

Se obține folosind opțiunea THROTTLE ce constă în executarea programului pas cu pas, după executarea fiecărei instrucțiuni făcându-se o pauză stabilită de utilizator.

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

În paralel cu executarea unui program în fereastra TRACE, se poate vizualiza conținutul variabilelor în fereastra WATCH.

Fereastra WATCH are două părți : în partea de sus se introduc variabile și expresii ce se doresc să fie vizualizate în timpul execuției programului, iar în partea de jos apar valorile de moment ale variabilelor. Deasemeni, în DEBUG se pot specifica puncte de întrerupere a programului, dependente de valorile variabilelor din fereastra respectivă, spre deosebire de punctele de întrerupere stabilite în TRACE care reprezintă puncte fixe ale programului, independente de valorile variabilelor.

Pentru a poziționa un astfel de punct, pe bara ce separă cele două părți se va amplasa un marcat cu mouse-ul sau deplasându-ne pe bara separatoare cu TAB , poziționându-ne cu ajutorul săgeților și apăsând SPACE.

Exemplu :



Dorim să calculăm suma primelor zece numere naturale.

```
CLEAR  
SET TALK OFF  
SUMA=0  
FOR I=1 TO 10  
SUMA=SUMA+I  
ENDFOR  
? "SUMA PRIMELOR ZECE NUMERE NATURALE ESTE: ", SUMA
```

Acest program dă rezultatul 55.

Introduceți o greșeală în program, de exemplu în loc de SUMA+I scrieți SUMA*I și depărtați programul după indicațiile de mai sus.

3.11 Instrumente WIZARD

Instrumentele wizard au fost introduse pentru a permite utilizatorilor mai puțin avansați să profite de avantajele programului. Un *wizard* este un instrument soft care ajută la efectuarea rapidă și comodă a operațiunilor de prelucrare a bazelor de date, oferă instrucțiuni, pune întrebări și în funcție de ceea ce i se răspunde execută procesul dorit. Visual Fox dispune de mai multe instrumente wizard dintre care: Table, Query, Report, Form, Label, Group/Total, Mail Merge.

Instrumentele Wizard se lansează astfel: din meniul Tools, alegând opțiunea Wizard apare o fereastră derulantă cu lista instrumentelor din care îl alegem pe cel dorit.

La lansarea unui wizard trebuie citite instrucțiunile și trebuie date răspunsurile la întrebările puse, apoi se alege butonul Next (sau Back dacă se

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

dorește întoarcerea la caseta anterioară) iar la ultima casetă se apasă butonul Finish.

Table wizard

- vă ajută să creați un tabel. Fereastra Table Wizard conține o listă a tabelelor tipice cu exemple de câmpuri frecvent utilizate. Selectați câmpurile dorite și ele vor fi incluse în noul tabel. Puteți modifica numele câmpurilor, tipul și lungimea lor precum și ordinea de sortare a datelor. Caseta de dialog 3 are 4 opțiuni și anume:
 - *Save table* – salvează tabelul;
 - *Save and browse for data entry* – salvează tabela și afișează ecranul pentru introducerea datelor ;
 - *Save and create an Autoscreen for data entry* – salvează tabelul și creează un ecran standard pentru introducerea datelor;
 - *Modify structure of table* – modifica structura tabelului.

Form Wizard

- vă ajută să creați un ecran pentru a gestiona informațiile din tabela dvs.
- la acest tip de Wizard se afișează mai întâi un ecran din care alegem tipul de ecran și anume:
 - *Form Wizard*-form asociat unui singur tabel;
 - *One to many Form Wizard*-forma ce are asociate două tabele (părinte și copil, relaționate) ;
- utilizați butoanele disponibile pentru a vă deplasa prin tabel și a efectua diverse operații. Butoanele pot avea atașate etichete text sau imagini explicative.
- în caseta 2 alegeti câmpurile ce dorîți să fie incluse în ecran. Un câmp se alege prin dublu click pe numele sau din lista *Avaible Fields* sau prin selectarea sa și apăsarea butonului *Add*. Butonul *Add All* include toate câmpurile din lista. Înlăturarea unui câmp din lista *Selected Fields* se face prin selectarea lui urmată de apăsarea butonului *Remove*.
- în caseta 3 se precizează ordinea de sortare a datelor.
- Caseta 4 folosește la selectarea stilului ecranului și a butoanelor declanșatoare.
- Caseta 5 va permite să stabiliți un titlu care va fi plasat în partea superioară a noului ecran și tipul de salvare:
 - *Save form for later use* – salvează ecranul pentru o întrebuițare ulterioară;
 - *Save and run form* – salvează și rulează ecranul;

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

- *Modify form with design tools* – modifică ecranul cu instrumentele de proiectare .

Report Wizard

- La acest tip de Wizard se afișează mai întâi un ecran din care alegem tipul de raport și anume:
 - *Group /Total Report Wizard* – raport cu grupuri și totaluri, care vă ajută să creați o structură care poate grupa înregistrările până la trei nivele de grupare și optional calculează totaluri parțiale și generale pentru câmpurile numerice;
 - *Multi Column Report Wizard* – vă asistă la crearea unui raport cu maxim trei coloane;
 - *Report Wizard* - raport .
- Opțiunile din casetele ce apar pe ecran sunt asemănătoare cu cele prezentate anterior.



3.12 Exerciții

1. Folosind tabelele create în lucrările anterioare, introduceți date în fiecare tabel, afișați structurile tabelei, testați pe rând fiecare comandă prezentată mai sus.
2. Având unul dintre fișierele create la lucrările anterioare:
 - creați o copie (folosind instrucțiunea COPY...);
 - creați un index (CDX);
 - comparați fișierul DBF după crearea indexului cu copia fișierului original;
 - ștergeți (DEL) fișierul index;
 - dați comanda USE cu fișierul DBF pentru care ați creat indexul.
3. Dați comanda PACK (fără să fie marcat pentru ștergere vreun articol) pentru unul din fișierele DBF. Urmăriți lungimea fișierului înainte și după executarea comenzi.
4. Pentru fișierul agenti, creați următoarele indexuri și afișați conținutul tabelei cu fiecare index dintre :
 - nume+prenume,
 - nume+număr de identificare (marca),
 - nume+data nașterii – descrescător,
 - nume+data angajării +salar (descrescător)
5. Scrieți procedurile pentru marcarea la ștergere, în tabela cu agentii firmei, a următoarele articole: agentii având același nume, agentii ce au numele asemănător cu unul introdus de la tastatura.
6. Testați comenziile REINDEX și PACK
7. Folosind unul dintre fișierele existente, testați comanda ZAP în ambele variante, pentru SET SAFETY ON și OFF
8. Folosind baza de date Biblioteca creată în lucrarea anterioară:
 - stabiliți relațiile dintre tabele.
 - stabiliți regulile de integritate referențială.
 - Închideți și redeschideți baza de date. Verificați dacă relațiile stabilite anterior sunt prezente.
9. Creați o bază de date numită Firma care să cuprindă tabelele Agenti, Clienti, Mărfuri, Tranzacții, Facturi cu structura de mai jos. Studiați tabelele și creați pentru fiecare indecșii necesari, apoi relaționați tabelele.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

CLIENTI

NR_CLIENT N (8,0)
DENUMIRE C (15)
TIP C (3) (SA, SRL,etc.)
LOCALITATE C (10)
JUDET C (2)
PRIVAT L

AGENTI

NR_AGENT N (4,0)
NUME C (10)
PRENUME C (10)
ORAS C (10)
JUDET C (2)
ANGAJARE D
NASTERE D
SEX C (1)
CAS L (casatorit,etc.)
SAL_BAZA N (6,0)

TRANZACT

NR_TRANZ N (6,0)
NR_CLIENT N (8,0)
NR_AGENT N (4,0)
COD_PRODUS N (6,0)
CANT N (4,0)
DATA_TRANZ D
NR_FACT N (6,0)
TIP C (1) (intrări, ieșiri)

FACTURI

NR_FACT N (6,0)
NR_TRANZ N (6,0)
DATA_EM D
ACH L (achitata sau nu)
DATA_ACH D
VALOARE N (10,0)

MARFURI

COD_PRODUS N (6,0)
DENUMIRE C (12)
TIP_MODEL C (10)
UN_MAS C (6)
PRĒT_UN N (8,0)
CANTITATE N (8,0)

10. Adăugați în tabelul AGENTI o coloana numerică PRIME și completați valorile după cum urmează: 100000 pentru cei din orașul Iași, 75000 pentru cei din restul județului, 60000 pentru ceilalți.
11. Ștergeți din tabelul MARFURI liniile cu cantitatea sub 10, sau prețul unitar <1500. Adăugați în tabelul FACTURI o coloana TVA și completați valorile cu 19% din coloana VALOARE.
12. Creati un fișier numit SALARIAT cu structura: nume, salar, impozit, rest_plata. Calculați impozitul pentru persoanele cu un salar mai mare decât o valoare pe care o introduceți folosind comanda ACCEPT. Pentru calculul impozitului creați o funcție numita IMPOZIT. (Exemplu: pentru salarii intre 500000-1500000 impozit este 105000 plus 28% din diferența dintre salar și baza de 500000).
13. Scrieți un program care să va permite introducerea datelor într-un fișier creat de dvs. folosind comenziile de intrare/ieșire învățate în aceasta

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

lucrare. Folosiți instrucțiunile de validare a datelor pentru câmpurile unde acest lucru este necesar.

14. Scrieți un program pentru calcularea mediilor studenților unei grupe formată din două semigrupe. Ca date de intrare aveți fișierul STUD.DBF cu structura :

NUME	C(15)
PRENUME	C15)
ID	N(3,0) && număr matricol
NOTĂ1	N(3,0)
NOTĂ2	N(3,0)
NOTĂ3	N(3,0)
NOTĂ4	N(3,0)

NOTĂ zero reprezintă absența la examenul respectiv.

- Ca rezultat se cere un fișier CLAS.DBF în care apar două câmpuri suplimentare nr_crt. și media.
- Articolele trebuie sortate în ordine crescătoare a semigrupelor și a mediilor, apoi în ordine crescătoare pentru nume, prenume și număr matricol.
- Mediile se vor calcula cu o funcție, după următoarea regulă: pentru o singură absentă, se împarte suma notelor la 3, altfel la 4.
- Atenție la folosirea literelor mari și mici și la eventualele spații de la începutul câmpurilor caracter.

15. Scrieți un program de calcul al salariilor agenților comerciali pentru luna precedentă celei curente. La salariul de bază se adăuga 3% din tranzacțiile "cumpărare" și respectiv 6% din cele de vânzare intermediate.

Impozitul se calculează astfel:

- se determină suma bruta ca fiind salbaza+sporuri=brut.
- pentru brut intre 500000 și 1000000-105000 + 10% din brut - 500000
- pentru brut intre 1000000 și 1500000-275000 + 25% din brut-1000000

16. Creați fișierul DATEPERS.DBF cu următoarele elemente:

NUME	C(10)
PRENUME	C(10)
NR_AGENT	N(4)
SALBAZAN(6)	
SPORURI	N(6) && procente din baza+comision
IMPOZIT	N(6)
RETINERI	N(6)

Ca rezultat se cere un fișier numit SALAR.DBF (tot cu o coloana nr_crt.) în care să apară liniile sortate crescător după nume, prenume, nr_agent.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

17. Scrieți un program care să creeze un fișier pentru facturile neachitate și numărul de zile de întârziere acumulate, grupate pe clienți, încasări și plăti.
18. Scrieți un program care să realizeze un joc asemănător cu spânzurătoarea. Cuvintele ce trebuie ghicite sunt stocate într-un fișier numit BAZA. Numărul de încercări poate fi 5, iar în loc de desenul cunoscut puteți afișa diverse mesaje.

Capitolul 4 Programarea orientată pe obiect-clase și obiecte în Visual Fox Pro

Mediul Visual Fox Pro este compatibil atât cu programarea structurată standard cât și cu Programarea Orientată pe Obiect (POO), programare ce reprezintă o abordare nouă, în care în loc să ne gândim la execuția unui program de la prima până la ultima linie de cod, va trebui să analizăm obiectele, care sunt componente de sine stătătoare ale aplicației, împreună cu proprietățile și metodele lor.

Un obiect reprezintă ansamblul de date și procedurile de prelucrare a acestora. În acest context, procedurile se numesc metode iar caracteristicile obiectului sunt proprietăți.

POO este o modalitate de a împacheta codul astfel încât să poată fi refolosit și întreținut mai ușor.

Primul nivel de împachetare se numește **clasă**. Clasa reprezintă definiția, într-un anumit limbaj de programare, a unui tip de obiecte, adică descrierea proprietăților și metodelor componente.

Clasele și obiectele sunt înrudite, dar nu sunt similare. Clasa conține informații despre cum trebuie să arate și să se comporte un obiect, deci determină caracteristicile acestuia.

Un obiect are proprietăți sau atribute determinate de clasa care stă la baza lui, proprietăți ce pot fi stabilite în faza de proiectare sau execuție.

De exemplu, în tabela de mai jos sunt prezentate câteva dintre proprietățile obiectului numit casetă de validare :

Proprietatea	Descriere
Caption	Textul descriptiv de lângă caseta de validare
Enabled	Specifică dacă utilizatorul poate selecta sau nu caseta de validare
ForeColor	Culoarea textului afișat drept titlu
Left	Pozitia marginii din stânga a casetei de validare
MousePointer	Felul în care arată indicatorul mouse-ului când se află deasupra casetei de validare
Top	Pozitia marginii de sus a casetei de validare
Visible	Specifică dacă pentru utilizator caseta de validare este vizibilă sau nu

Metode și evenimente asociate unui obiect

Fiecare obiect poate recunoaște și poate reacționa la unul sau mai multe evenimente.

Un **eveniment** este o acțiune specifică și predeterminată, inițiată de utilizator sau de sistem. În Visual Fox Pro acțiunile utilizatorilor ce pot declanșa evenimente sunt mișcarea mouse-ului, apăsarea unei taste, etc. Evenimentele declanșate de sistem sunt inițializarea unui obiect sau întâlnirea unei linii de cod ce generează o eroare.

Metodele sunt procedurile atașate unui obiect. Metodele diferă de procedurile obișnuite deoarece sunt legate indisolubil de obiect și sunt apelate altfel decât celelalte proceduri.

Evenimentele pot avea metode asociate. De exemplu, dacă scrieți cod pentru metoda asociată evenimentului Click, codul respectiv se execută la apariția evenimentului click.

Tabelul de mai jos prezintă câteva evenimente asociate unei casete de validare:

Eveniment	Descriere
Click	Utilizatorul execută click în caseta de validare
GotFocus	Utilizatorul selectează caseta de validare printr-un click de mouse sau prin apăsarea tastei TAB
LostFocus	Utilizatorul selectează alt control
Refresh	Valoarea casetei de validare este actualizată pentru a reflecta modificările survenite în cadrul sursei de date asociate
SetFocus	Caseta de validare își atribuie focusul (devine ținta intrărilor) ca și cum utilizatorul ar fi apăsat tasta TAB

Clasele

Toate proprietățile, metodele și evenimentele asociate unui obiect sunt specificate în cadrul definiției clasei. În plus, clasele au o serie de caracteristici ce permit scrierea de cod care este ușor de întreținut și poate fi reutilizat.

Caracteristicile claselor sunt :

- **Încapsularea** – implică împachetarea codului în cadrul obiectului și abstractizarea acestuia. De exemplu, proprietățile ce determină elementele unui casetă cu lista și codul ce se execută atunci când selectați un element din listă pot fi încapsulate într-un singur control ce se adaugă la un formular.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

- **Subclasele** – sunt derivate dintr-o clasa existentă. Subclasele moștenesc toate funcționalitățile clasei părinte, dar pot avea și proprietăți noi, pe care programatorul le poate seta, reutilizând astfel codul deja existent.
- **Moștenirea** este caracteristica ce permite ca o modificare efectuată asupra unei clase să se reflecte în toate clasele definite pe baza ei, actualizarea automată ducând la economisirea timpului și a efortului dvs.

În Visual Fox Pro clasele pot fi de tip control sau container, după cum se arată în lista următoare:

- **Controale:** Check Box, Combo Box, Command Button, Edit Box, Header, Hyperlynk, Image, Label , Line, List Box, OLE Bound Control, OLE Container Control, Shape, Spinner, Text Box, Timer
- **Containere:** Container, Form Set, Form ,Grid, Column, Page Frame, Page, Toolbar, Option Button Group, Command Button Group

Clasele containere pot să conțină alte obiecte, permitând accesul la acestea. De exemplu, în cazul în care creați o clasă container care cuprinde o casetă listă și două butoane de comandă, după care adăugați unui formular un obiect bazat pe această clasă, fiecare obiect poate fi manipulat individual, atât la proiectare cât și în timpul execuției.

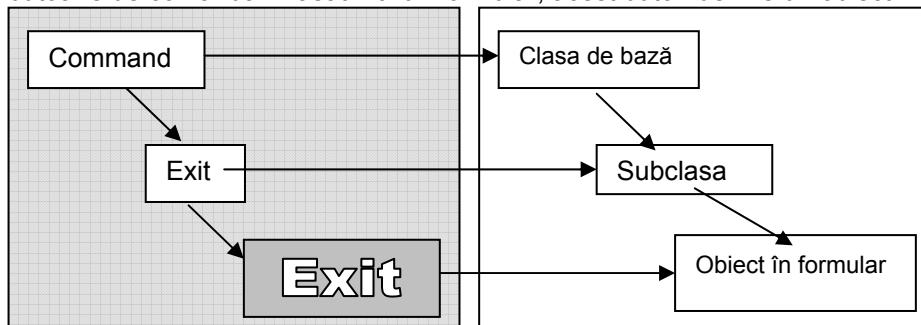
Toate clasele recunosc următorul set minimal de evenimente:

Eveniment	Descriere
Init	Apare la crearea unui obiect
Destroy	Apare la descărcarea unui obiect din memorie
Error	Apare de câte ori se semnalează o eroare

Toate clasele recunosc următorul set minimal de proprietăți:

Proprietate	Descriere
Class	Tipul clasei
BaseClass	Clasa de bază din care provine (Form, CommandButton, etc.)
ClassLibrary	Biblioteca de clase în care este stocată
ParentClass	Clasa din care a fost derivată (dacă derivă dintr-o clasă de bază, ParentClass este aceeași cu BaseClass)

Exemplu : plecând de la clasa de bază CommandButton putem crea un buton care să execute ieșirea din formular, cu eticheta Exit, ca o subclasă a clasei butoane de comandă. Plasat într-un formular, acest buton devine un obiect.



4.1 Crearea claselor

Se poate face din cadrul gestionarului de proiecte, selectând pagina Classes și apoi New sau din meniul File, opțiunea New, Class, New File sau folosind comanda :



CREATE CLASS ClassName | ? [OF ClassLibraryName1 | ?] [AS cBaseClassName [FROM ClassLibraryName2]] [NOWAIT]

Modificarea unei clase va afecta toate subclasele și obiectele bazate pe acea clasă și se face prin selectarea clasei dorite și alegerea opțiunii Modify.

Proiectantul de clase

Proiectantul de clase se deschide atunci când este indicată clasa pe care se va baza noua clasă creată și biblioteca în care aceasta va fi stocată. Interfața proiectantului de clase ne permite vizualizarea și modificarea proprietăților clasei în fereastra Properties. Deasemeni se poate introduce sau modifica codul asociat metodelor necesare.

Dacă noua clasă se bazează pe una de tip container, se pot adăuga controale (butoane, grile, imagini, etc.).

Unei clase existente i se pot adăuga proprietăți noi astfel:

- Din meniul Class, selectați New Property;
- În caseta New Property introduceți numele proprietății;
- Specificați vizibilitatea (Public, Protected, Hidden) ;
- Dați click pe Add.

Într-un mod asemănător se poate adăuga și o metodă nouă unei clase.

Atunci când adăugați o proprietate nouă, valoarea ei implicită este .F.

Pentru a specifica o altă valoare, în fereastra Properties, în fișa Other, dați click pe proprietatea dorită și introduceți noua valoare.

Atribuirea unei pictograme pentru noua clasă în vederea afișării pe bara de instrumente se face prin selectarea opțiunii Class Info din meniu Class (trebuie să fiți în proiectantul de clase). În caseta Class Info introduceți numele și calea pentru fișierul.bmp din caseta Toolbar Icon.

Adăugarea claselor la formulare se face astfel :

1. Din meniu Tools, alegeti opțiunea Options, apoi alegeti fișa Controls;
2. Selectați Visual Class Libraries și alegeti Add;
3. Din caseta de dialog Open alegeti o biblioteca de clase pe care sa o adăugați și selectați Open.

Definirea claselor prin program se face cu comanda :



```
DEFINE CLASS ClassName1 AS ParentClass [OLEPUBLIC]
[[PROTECTED | HIDDEN] PropertyName1, PropertyName2...]
[Object.]PropertyName = eExpression...] [ADD OBJECT
[PROTECTED] ObjectName AS ClassName2
[NOUNIT][WITH cPropertylist]]...
[[PROTECTED | HIDDEN] FUNCTION | PROCEDURE Name
[NODEFAULT]cStatements [ENDFUNC | ENDPROC]]...
ENDDEFINE
```

Exemplu : Crearea unui set de butoane de navigare pentru vizualizarea înregistrărilor dintr-o tabelă.



Butoanele de navigare vor executa următoarele comenzi:

- poziționare pe prima înregistrare din tabela – navtop;
- poziționare pe înregistrarea anterioara celei curente-navprior;
- poziționare pe înregistrare următoare celei curente – navnext;
- poziționare pe ultima înregistrare din tabela – navbottom.

Butoanele vor avea caracteristici comune, de aceea merită să creăm o clasă pentru toate butoanele de navigare. Clasa părinte este numită Navbutton și va fi definită ulterior.

Odată definită clasă părinte, subclasele ce urmează definesc funcționalitatea și aspectul specific fiecărui buton. Va fi creată o clasă container,

numită vcr, în care va fi adăugat fiecare buton. Containerul va fi apoi adăugat formularelor pentru a se putea naviga în cadrul tabelei.



Exemplu:

Scriți un program numit navclass.prg

***Definirea clasei NAVBUTTON**

*Define class Navbutton as CommandButton
Height=25
Width=25
TableAlias = ''*

*Procedure click
if not empty(this.TableAlias)
 select (this.TableAlias)
endif
endproc*

*Procedure refreshform
 screen.ActiveForm.Refresh
Endproc*

Enddefine

*** Definirea butonului Top**

*Define class Navtop as Navbutton
caption='T'*

*Procedure click
dodefault()
go top
this.refreshform
endproc
Enddefine*

*** Definirea butonului Prior**

Define class Navprior as Navbutton
caption='P'

```
Procedure click
dodefault()
skip -1
if bof()
go top
endif
this.refreshform
endproc
Enddefine
```

*** Definirea butonului Next**

Define class Navnext as Navbutton

```
caption='N'
Procedure click
dodefault()
skip 1
if eof()
go bottom
endif
this.RefreshForm
endproc
Enddefine
```

*** Definirea butonului Bottom**

Define class Navbottom as Navbutton
caption='B'

```
Procedure click
dodefault()
go bottom
this.refreshform
endproc
Enddefine
```

*** Definirea clasei vcr**

```
define class vcr as container
height=25
width=100
left=50
top=3
add object cmdtop as navtop with left = 0
add object cmdprior as navprior with left = 25
add object cmdnext as navnext with left = 50
add object cmdbottom as navbottom with left = 75
```

```
procedure settable(cTableAlias)
if type("cTableAlias")='C'
this.cmdtop.TableAlias=cTableAlias
this.cmdprior.TableAlias=cTableAlias
this.cmdnext.TableAlias=cTableAlias
this.cmdbottom.TableAlias=cTableAlias
endif
enddefine
```

***Adăugarea controlului vcr la o clasa formular**

```
define class navform as form
add object ovcr as vcr
enddefine
```

4.2 Crearea formularelor

Formularele (sau forme, machete, ecrane) sunt interfețe grafice ce oferă utilizatorului posibilitatea de a acționa asupra tabelelor de date prin operații de adăugare, modificare, ștergere sau consultare a informațiilor stocate. Formularele sunt compuse din obiecte ce pot răspunde la evenimente generate de utilizator sau sistem, astfel încât utilizatorul să-și poată îndeplini cât mai intuitiv și ușor sarcinile.

Un formular nou se creează cu ajutorul proiectantului de formulare (*Form Designer*). Acesta poate fi accesat alegând din meniul *File* opțiunea *New Form* sau folosind comanda **CREATE FORM**.

Fiecare formular include un mediu de date. *Mediul de date* este un obiect ce cuprinde tabelele și vederile cu care interacționează formularul, precum și relațiile dintre tabele. Mediul de date poate fi salvat împreună cu formularul, astfel încât ori de câte ori respectivul formular va fi accesat, mediul de date va deschide tabelele sau vederile asociate, iar la ieșirea din formular acestea vor fi automat închise.

Proprietatea *ControlSource* a controalelor va fi populată, făcându-se

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

automat legătura între sursa de date și ceea ce va fi afișat pe ecran.

Pentru a accesa fereastra de proiectare a mediului de date alegem din meniu opțiunea **View**, **Data Environment**, apoi click dreapta și **Add**. În caseta de dialog **Open** alegeti tabela dorită, repetând operația de adăugare până ce lista de tabele este epuizată.

Cele mai importante proprietăți ce trebuie setate pentru mediul de date sunt:

Proprietate	Descriere	Valoare implicită
AutoCloseTables	Controlează închiderea automată a tabelelor la eliberarea formularului din memorie	Adevărat(.T.)
AutoOpenTables	Controlează deschiderea automată a tabelelor la rularea formularului	Adevărat(.T.)
InitialSelectedAlias	Indică tabela selectată la rularea formularului	Dacă nu se specifică, în momentul execuției este selectat primul cursor adăugat în DataEnvironment

Eliminarea unei tabele din cadrul mediului de date se face prin selectarea tablei urmată de selectarea opțiunii **Remove** din meniul **DataEnvironment**.

Dacă pentru tabelele ce au fost adăugate în mediul de date existau relații permanente, acestea se păstrează și sunt vizibile.

Dacă nu există relații persistente între table, acestea se pot crea prin tragerea unui câmp din tabela primară peste eticheta de index corespunzătoare din tabela asociată. Atunci când se stabilește o relație în cadrul mediului de date, între tabele va apărea o linie care reprezintă relația, proprietățile acesteia putând fi setate din fereastra **Properties**.

După ce am definit mediul de date asociat unui formular va trebui să ii adăugam obiectele necesare.

Controalele dintr-un formular pot fi de două tipuri : controale care sunt asociate unor surse de date și controale ce nu au date asociate. Atunci când utilizatorul interacționează cu controalele asociate datelor, valorile introduse de el sunt stocate în sursa de date.

Proprietățile ce determină asocierea cu datele sunt **ControlSource** pentru un control grilă sau **RecordSource** pentru celelalte. Dacă proprietatea **ControlSource** nu este configurață, valoarea introdusă de utilizator sau cea pe care o alege din cadrul controlului este păstrată ca valoare a proprietății și nu va fi salvată pe disc sau în memorie.

4.3 Utilizarea controalelor predefinite



Butoane de opțiune

Grupurile de butoane de opțiune sunt containere ce cuprind butoane de opțiune. Ele permit utilizatorilor să aleagă doar una dintre opțiunile puse la dispoziție în loc să introducă direct datele. De exemplu, aceste butoane pot fi folosite în cazul în care dorim ca o listă să fie afișată pe ecran sau trimisă către o imprimantă. Implicit în grupul de butoane de opțiune sunt introduse două butoane.

Prin modificarea proprietății **ButtonCount** se poate specifica numărul de butoane dorit.

Proprietatea **Value** indică butonul care a fost selectat. Dacă utilizatorul alege butonul 3, atunci **Value** va avea valoarea 3.

Dacă proprietatea **ControlSource** a grupului este un câmp de tip caracter, sau dacă proprietății **Value** îi se atribuie o valoare de tip caracter înainte de rularea formularului, proprietatea **Value** este dată de titlul butonului selectat.

Configurarea proprietăților se poate face manual pentru tot grupul de butoane sau pentru fiecare buton în parte din fereastra **Properties**, sau în timpul execuției, indicând numele butonului și valoarea respectivă pentru proprietatea dorită.

Exemplu: considerăm că avem un grup de două butoane prin setarea cărora dorim să stabilim dacă un medicament se prezintă sub forma de tablete sau fiole. Va trebui să setăm proprietatea **Caption** a butonului, în fereastra de proprietăți sau să scriem codul asociat.



`ThisForm.optgr.Button(1).Caption="Tablete"`

`ThisForm.optgr.Button(2).Caption="Fiole"`

unde-`ThisForm` este forma activă în care se lucrează

-`Optgr` este numele grupului de butoane

-`Button` este butonul selectat

Configurarea proprietăților tuturor butoanelor dintr-un grup se poate face folosind metoda **SetAll** a grupului.

Exemplu:



`ThisForm.optgr.SetAll("Enabled",.F.,"OptionButton")`

va dezactiva toate butoanele din grup.

Pentru a determina care dintre butoane este selectat la un moment dat se folosește proprietatea **Value** a grupului.

Butoanele de opțiune pot fi folosite și pentru păstrarea informațiilor într-

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

o tabelă, prin salvarea proprietății **Caption**. Pentru aceasta va trebui să :

- Atribuim un sir vid proprietății **Value** a grupului de butoane;
- Atribuim Proprietății **ControlSource** a grupului de butoane un câmp de tip caracter dintr-o tabelă.



Casete listă și casete cu listă derulantă

Casetele tip listă și casetele tip listă derulantă (casete combinate, pentru care proprietatea **Style** are valoarea 2 – **DropDownList**) oferă utilizatorului liste cu cuprind informații ce pot reprezenta valori dintr-un tabel sau opțiuni predefinite.

Într-o caseta tip listă, în orice moment, sunt vizibile mai multe elemente ale listei.

Într-o caseta derulantă este vizibil doar un singur element. Utilizatorul poate executa click pe butonul săgeata în jos pentru a afișa toate elementele listei.

Proprietăți și metode ale unei liste :

Proprietate	Descriere
ColumnCount	Numărul de coloane din caseta listă
ControlSource	Păstrează valoarea aleasă de utilizator din cadrul listei
Multiselect	Indică dacă utilizatorul poate selecta mai multe elemente la un moment dat
RowSource	Sursa de proveniență a elementelor afișate în listă
RowSourceType	Indică tipul pentru RowSource (o valoare, o tabela, o interogare, etc.)
Metode	Descriere
AddItem	Adaugă un element la o listă pentru care proprietatea RowSourceType este 0
RemoveItem	Elimină un element dintr-o listă pentru care proprietatea RowSourceType este 0
Requery	Actualizează lista dacă valorile din RowSourceType s-au modificat

Alegerea tipului de date pentru o casetă listă sau casetă cu listă derulantă se face prin setarea proprietății **RowSourceType** și prin atribuirea unei surse de date, completând valoarea corespunzătoare pentru proprietatea **RowSource**.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

RowSourceType	Sursa elementelor listei
0	Nici una. Elementele listei se adaugă prin program
1	Valoare
2	Alias
3	Instrucțiune SQL
4	Interrogare (Query)(.qpr)
5	Matrice
6	Câmpuri dintr-o tabelă
7	Fisiere
8	Structură
9	Lista pop-up (inclusă pentru compatibilitate cu versiunile anterioare)



Exemple :

Dacă **RowSourceType** este 0, elementele listei se pot adăuga folosind metoda **AddItem**

ThisForm.list1.RowSourceType=0

ThisForm.list1.AddItem("Primul element")

ThisForm.list1.AddItem("Al doilea element")

ThisForm.list1.AddItem("Al treilea element")

Metoda **Remove Item** permite eliminarea elementelor din listă.

ThisForm.list1.RemoveItem(3) – pentru eliminarea celui de al treilea element

- Dacă **RowSourceType** este 1 se pot indica pentru proprietatea **RowSource** mai multe valori care să fie afișate în cadrul listei. Elementele trebuie separate prin virgulă iar dacă acestea vor fi introduse prin program vor trebui încadrate între ghilimele.

ThisForm.list1.RowSourceType=1

ThisForm.list1.RowSource="Primul,Al doilea, Al treilea"

- Dacă **RowSourceType** este 2 se pot insera valori din câmpurile tabelei deschise, în funcție de setarea pentru **ColumnCount**.

Dacă **ColumnCount** este 0 sau 1 se afișează primul câmp al tabelei, dacă este 3, lista afișează valorile din primele trei câmpuri. Pentru a afișa câmpurile într-o altă ordine decât cea din tabela setați **RowSourceType** pe 3 sau pe 6.

- Dacă **RowSourceType** este 3 trebuie să includeți o instrucțiune SQL SELECT în proprietatea RowSource. Dacă instrucțiunea va fi scrisă prin program, ea va trebui inclusă între ghilimele. Este de preferat ca rezultatele interrogării să fie plasate într-un cursor pentru a nu se afișa

rezultatul într-o fereastră Browse.

- Dacă **RowSourceType** este 4 caseta poate fi populată cu rezultatul unei interogări salvate într-un fișier cu extensia “.qpr”, caz în care proprietății Row Source i se atribuie fișierul.qpr.
`ThisForm.List1.RowSource="numefisier.qpr"`
- Dacă **RowSourceType** este 5 atunci lista este populată cu elementele unei matrici.
- Dacă **RowSourceType** este 6 atunci lista este populată cu câmpurile din structura fișierului dorit specificate în RowSource și delimitate prin virgulă.
- Dacă **RowSourceType** este 7 atunci lista cuprinde fișierele din directorul curent, cu opțiunea că pot fi alese și alte unități de disc.
- Dacă **RowSourceType** este 8 atunci lista cuprinde câmpurile specificate în tabela precizată la **RowSource**.

Se poate oferi utilizatorului posibilitatea de a alege o opțiune din listă, pentru care să fie afișate o serie întreagă de alte informații. De exemplu, dacă lista este populată cu numele pacienților tratați de un medic și dorim să afișăm toate informațiile legate de pacient, putem proceda astfel, în funcție de sursa de date a formularului:

RowSourceType	Selectarea înregistrărilor corespunzătoare
2-Alias 6-Câmpuri	Când utilizatorul selectează o valoare din listă, indicatorul de înregistrare este fixat automat pe înregistrarea dorită. Se lansează comanda ThisForm.Refresh în evenimentul InteractiveChange al listei pentru afișarea noilor valori în celelalte elemente ale formei
0-Nici una 1-Valori 3-Instrucțiune SQL 4-Interrogare (qpr)	În cadrul evenimentului InteractiveChange selectați tabela ce conține înregistrarea cu valoarea dorită, apoi căutați valoarea respectivă. De exemplu dacă RowSource conține numele de identificare pentru pacienți, se poate folosi codul următor <code>Select pacienți</code> <code>Locate for This.Value=pacient_id</code> <code>Thisform.Refresh</code>



Casete de validare

Casetele de validare sunt folosite pentru a permite utilizatorului să specifice o stare de tip logic (True sau False, activat sau dezactivat, etc).

Totuși există cazuri în care nu se poate evalua cu precizie starea dorită, motiv pentru care există patru stări posibile pentru caseta de validare și anume :

Forma afișată Proprietatea Value

... <input type="checkbox"/> Check1	0 sau.F.	Proprietatea Value a casetei de validate indică tipul de date corespunzător ultimei atribuiri efectuate. Dacă s-a setat valoarea .T. sau .F. tipul de date este Logical, și nu se va schimba decât atunci când veți atribui proprietății o valoare numerică.
... <input checked="" type="checkbox"/> Check2	1 sau.T.	
... <input checked="" type="checkbox"/> Check3	2	
... <input checked="" type="checkbox"/> Check4	Null	

Dacă se asociază proprietății **Control/Source** a unei casete de validare un câmp de tip logic dintr-o tabelă, caseta va fi :

- bifată dacă valoarea înregistrării curente este .T. ;
- nu va fi bifată dacă valoarea înregistrării curente este .F. ;
- va avea un semn pe fond cenușiu dacă valoarea înregistrării curente este .null.



Caseta tip text

Caseta tip text este controlul de bază în care utilizatorul poate să introducă sau să modifice datele păstrate într-un câmp al unei tabele, câmp ce nu este de tip memo.

Configurarea sau modificarea prin program a textului afișat în casetă se face prin setarea proprietății **Value** a acesteia.

Dacă proprietatea **Control/Source** este specificată, valoarea afișată în caseta text este stocată în proprietatea **Value** a acesteia și în variabila sau câmpul specificat în **Control/Source**.

Validarea datelor dintr-o caseta tip text se face prin scrierea de cod asociat metodei **Valid**. Dacă metoda **Valid** întoarce valoarea .F. va fi afișat mesajul "Invalid Input" sau propriul dvs. mesaj specificat în codul evenimentului **Valid** (se poate folosi funcția **Messagebox()** sau comanda **Wait Wind** 'mesajul dvs.').

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Exemplu: dorim să introducem într-o casetă de tip text un număr mai mare decât 100. Pentru a valida datele introduse de utilizator inseram codul asociat metodei **Valid** astfel:

```
If val(This.Value) < 100  
    = Messagebox("Introduceți o valoare mai mare decât 100",1)  
    Return 0  
Endif
```

Pentru a selecta textul atunci când caseta text devine ținta intrărilor atribuiți valoarea **.T.** proprietății **SelectOnEntry**.

Proprietatea **InputMask** determină caracteristicile textului introdus în casetă. De exemplu dacă **InputMask** are valoarea 999,999.99 se vor introduce în casetă doar numere mai mici de un milion, cu două zecimale. Virgula și punctul vor fi afișate în casetă înainte de introducerea datelor.

Pentru câmpuri de tip logic, dacă se dorește ca utilizatorul să poată introduce "Y" (Da) și "N" (Nu) dar nu și "T" sau "F" atribuiți valoarea "Y" proprietății **InputMask**.

Dacă se dorește acceptarea introducerii unor valori fără ca acestea să apară în clar (de exemplu la introducerea unor parole), se atribuie proprietății **PasswordChar** valoarea "*" sau orice alt caracter generic.

Pentru a introduce date calendaristice vor fi setate următoarele proprietăți:

Proprietate	Descriere
Century	Indică dacă sunt afișate sau nu primele două cifre ale anului
DateFormat	Formatează datele de intrare într-unul dintre cele 15 formate de dată predefinite
StrictDateEntry	Dacă este 0 permite introducerea datelor într-un format mai flexibil decât cel implicit 99/99/99.

Cele mai importante proprietăți pentru casetele tip text sunt :

Proprietate	Descriere
Alignment	Setează tipul de aliniere a valorilor introduse în caseta (stânga, dreapta, centru, automat). Alinierarea depinde de tipul de dată introdus.
ControlSource	Câmpul tableei sau variabila a cărei valoare este afișată

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

InputMask	Indică regulile de introducere a datelor
SelectOnEntry	Stabilește dacă la primirea focusului conținutul casetei va fi selectat automat
TabStop	Indică dacă utilizatorul poate ajunge la caseta apăsând repetat tasta TAB. Dacă TabStop este .F. , ea poate fi totuși selectată efectuând click pe ea.



Caseta de editare

Casetele de editare permit afișarea și editarea unui text provenit din câmpuri lungi de caractere sau câmpuri memo. Aceste casete asigură saltul automat al textului pe rândul următor și posibilitatea de deplasare în cadrul textului folosind săgețile.

Casetele de editare, ca și casetele text au proprietăți ce permit lucrul cu textul selectat și anume: **SelLength**, **SelStart** și **SelText**



De exemplu, putem selecta primul cuvânt dintr-o caseta astfel:

`Form1.text1.SelStart=0`

`Form1.text1.SelLength= AT(" ",Form1.Text1.text)-1`

Cele mai importante proprietăți ce pot fi configurate pentru casetele de editare sunt :

Proprietate	Descriere
AllowTabs	Stabilește dacă este permisa utilizarea tastei TAB în caseta de editare (caz în care trebuie verificat dacă se poate trece la controlul următor cu CTRL+TAB)
HideSelection	Stabilește dacă textul din caseta de editare este selectat în mod vizibil atunci când caseta nu deține focusul
ReadOnly	Stabilește dacă textul din caseta poate fi modificat sau nu
ScrollBars	Indică prezenta sau absența barelor de defilare



Casetele combinate

Acest tip de casetă reunește funcționalitățile casetelor de tip listă și a celor de tip text.

Ele pot fi de două feluri: casete combinate derulante și liste derulante, funcție de specificația asociată proprietății **Style** a controlului.

Cele mai importante proprietăți ce pot fi configurate pentru casetele combinate sunt :

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Proprietate	Descriere
ControlSource	Indică câmpul în care va fi păstrată valoarea aleasă sau introdusă de utilizator
DisplayCount	Numărul maxim de elemente afișate în listă
InputMask	Tipul valorilor ce pot fi introduse
IncrementalSearch	Indică dacă controlul încearcă să găsească un element din listă care să corespundă cu literele deja introduse
RowSource	Sursa datelor din listă
RowSourceType	Indică tipul sursei casetei combinate. Valorile sunt similare cu cele de la caseta tip listă
Style	Indica dacă caseta este o casetă combinată derulantă sau listă derulantă.

Exemplu :



Dorim să adăugam noi valori la elementele afișate într-o casetă combinată cu listă derulantă. La evenimentul **Valid** vom scrie următoarea secvență de cod care, înainte de adăugarea elementului, să verifice dacă acesta nu există deja în listă.

Elexista =.F. (presupunem ca valoarea nu există)

```

For I=1 to This.ListCount
If This.List(i)= This.Text
Elexista=.T.
Exit
Endif
Endfor
If !Elexista
This.AddItem (This.text)
Endif

```



Casetele de incrementare/decrementare (Spinner)

Se folosesc pentru a permite utilizatorilor să introducă valori dintr-o plajă anterior stabilită.

Intervalul de valori se specifică prin setarea proprietăților **KeyboardHighValue** și **SpinnerHighValue** (valoarea maximă posibilă) și **KeyboardLowValue** și **SpinnerLowValue** (valoarea minimă posibilă).

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Deși controlul **Spinner** este configurat pentru valori numerice, el poate fi utilizat în combinație cu o casetă de tip text pentru a permite utilizarea altor tipuri de date.

Exemplu : Pentru a parurge un interval de date calendaristice, se va dimensiona controlul astfel încât să fie vizibile doar butoanele și plasam caseta text lângă el. Atribuim o dată calendaristică proprietății **Value** a casetei text iar în codul evenimentelor **UpClick** și **DownClick** ale **Spinner**-ului incrementăm sau decrementăm data.

Cele mai importante proprietăți ce pot fi configurate pentru **Spinner** sunt :

Proprietate	Descriere
Interval	Pasul de incrementare/decrementare
KeyboardHighValue	Valoarea maximă ce poate fi introdusă în caseta text a controlului
KeyboardLowValue	Valoarea minimă ce poate fi introdusă în caseta text a controlului
SpinnerHighValue	Cea mai mare valoarea ce poate fi afișată de control când se apasă butonul săgeata sus
SpinnerLowValue	Cea mai mică valoarea ce poate fi afișată de control când se apasă butonul săgeata jos



Buton de comanda



Grup de butoane de comanda

Butoanele de comanda permit executarea unor acțiuni. De obicei codul asociat acțiunii este specificat în evenimentul **Click**.

Cele mai importante proprietăți ce pot fi configurate pentru butoanele de comanda sunt :

Proprietate	Descriere
Cancel	Indică dacă se execută sau nu codul asociat evenimentului Click la apăsarea tastei ESC
Caption	Textul afișat pe buton
DisablePicture	Fișierul de tip imagine (bmp) afișat la dezactivarea butonului
DownPicture	Fișierul de tip imagine (bmp) afișat la activarea butonului
Enabled	Dacă butonul poate fi selectat sau nu
Picture	Fișierul de tip imagine (bmp) afișat pe buton

Facultatea de Automatică și Calculatoare Iași **Baze de date – lucrări practice**

Dacă se dorește folosirea unui grup de butoane de comandă înseamnă ca ele au caracteristici comune și codul asociat se poate scrie într-o singură procedură (de exemplu codul pentru **Click**). Proprietate **Value** a grupului indică pe care dintre butoane s-a dat click.

Dacă este scris cod pentru **Click** special pentru unul dintre butoanele din grup, atunci la declanșarea evenimentului **click** se va executa acest cod și nu cel general pentru grup.

Cele mai importante proprietăți ce pot fi configurate pentru grupurile de butoane de comandă sunt :

Proprietate	Descriere
ButtonCount	Numărul de butoane din grup
BackStyle	Tipul de fundal (transparent sau opac)



Controlul imagine

Permite afișarea imaginilor stocate într-un fișier de tip “bmp” pe un formular.

Proprietățile acestui control pot fi schimbate prin cod în timpul execuției astfel încât imaginea poate fi diferită de la un caz la altul.

Cele mai importante proprietăți ce pot fi configurate pentru controlul imagine sunt :

Proprietate	Descriere
Picture	Imaginea (fișierul bitmap) atașat
BorderStyle	Există sau nu chenar vizibil
Stretch	0-Clip - porțiunile imaginii ce depășesc dimensiunile controlului nu sunt afișate 1-Isometric – sunt păstrate dimensiunile originale ale imaginii 2- Stretch – imaginea este scalată corespunzător dimensiunilor controlului



Controlul etichetă

Diferă de caseta text prin următoarele : nu are sursă de date, nu poate fi editat direct din interiorul său, nu se poate ajunge la el prin apăsarea tastei TAB. Proprietățile **Caption** și **Visible** pot fi modificate prin program.

Cele mai importante proprietăți ce pot fi configurate pentru controlul etichetă sunt :

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Proprietate	Descriere
Caption	Textul afișat drept etichetă
AutoSize	Dacă dimensiunea etichetei este ajustată funcție de cea a textului din Caption
BackStyle	Dacă eticheta este opacă sau transparentă
WordWrap	Dacă textul afișat continuă pe mai multe rânduri



Forme geometrice și linia

Aceste controale ajută la gruparea vizuală a elementelor formularului și pentru înfrumusețarea lui.

Cele mai importante proprietăți ce pot fi configurate pentru forme geometrice sunt :

Proprietate	Descriere
Curvature	Valoare cuprinsă între 0 (unghi de 90 grade) și 99 (cerc sau oval)
FillStyle	Modelul de umplere al formei geometrice
SpecialEffect	Dacă forma este plană sau tridimensională. Are efect doar pentru Curvature=0

Cele mai importante proprietăți ce pot fi configurate pentru controlul linie sunt :

Proprietate	Descriere
BorderWidth	Grosimea liniei în pixeli
LineSlant	Direcția liniei în cazul în care nu este orizontală sau verticală (valorile valabile pot fi / sau \)



Controlul de tip grilă

Grila este un obiect de tip container, care poate conține mai multe coloane, ce pot avea antet și controale, fiecare cu propriul set de proprietăți, evenimente și metode.

Grila permite prezentarea și manipularea liniilor și coloanelor de date într-un formular sau o pagină și de cele mai multe ori este folosită pentru afișarea datelor din tabelele relaționate de tip una la mai multe.

Cele mai importante proprietăți ce pot fi configurate pentru controlul grilă sunt :

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Proprietate	Descriere
ChildOrder	Cheia externă a tableei fiu care este legată de cheia primară a tablei părinte
ColumnCount	Numărul de coloane: Dacă este -1 grila va avea atâtea coloane câte câmpuri există în sursa de înregistrări
LinkMaster	Tabela părinte a înregistrărilor fiu afişate în grilă
RecordSource	Datele ce vor fi afişate în grilă
RecordSourceType	Proveniența datelor din grilă (tabelă, alias, etc)

Cele mai importante proprietăți ce pot fi configurate pentru coloanele dintr-o grilă sunt :

Proprietate	Descriere
ControlSource	Datele ce vor fi afişate în coloană (câmp din tabele)
Sparse	Dacă este .T. – controalele grilei sunt afişate doar când celula respectivei coloane este selectată
CurrentControl	Stabilește care dintre controalele grilei este activ



Cadre de pagină

Un cadru de pagină este un obiect container ce poate conține mai multe pagini. La rândul lor, paginile pot conține controale.

Cele mai importante proprietăți ce pot fi configurate pentru cadrele de pagină sunt :

Proprietate	Descriere
Tabs	Indică dacă fișele paginilor sunt vizibile sau nu
TabStyle	Indică dacă foile sunt toate de aceeași dimensiune și dacă toate împreună au lățimea cadrului de pagină
PageCount	Numărul de pagini din cadrul de pagină

4.4 Generatorul de rapoarte

Rapoartele reprezintă un ansamblu de informații construite pe baza datelor din unul sau mai multe tabele, informații ce urmează a fi afișate pe ecranul monitorului, tipărite la imprimantă sau într-un fișier de tip text. Un raport este format dintr-un ansamblu de elemente care conlucrează la selectarea și furnizarea către utilizator a unui set de date într-o formă cat mai concisă, clară și placută.

Raportul se construiește în mediul VisualFoxPro cu ajutorul generatorului de rapoarte, folosind una din metodele :



Comanda CREATE REPORT <nume raport>

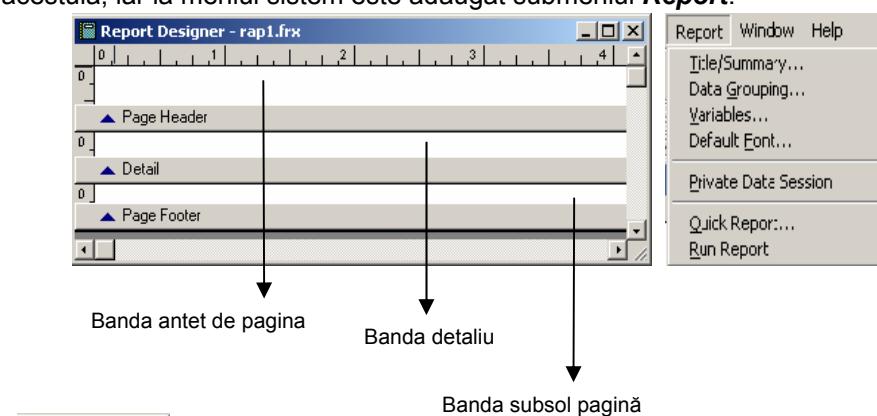
sau din meniul **File** alegem opțiunea **NEW** și apoi **Report**, precizând deasemeni numele raportului și calea unde acesta va fi salvat.

Modificarea unui raport existent se face folosind comanda :

MODIFY REPORT <nume raport>

sau din meniul **File** alegem opțiunea **Open** și apoi selectăm raportul dorit.

Odată lansat generatorul de rapoarte, pe ecran se afișează fereastra de lucru a acestuia, iar la meniul sistem este adăugat submeniu **Report**.



După cum se vede în figură, inițial orice raport are în componentă benzile cu semnificațiile menționate.

Pentru a putea configura elementele raportului, ne este necesară bara Report Control. Dacă acesta nu este vizibilă, va trebui să o activăm din meniul View, Toolbars, Report Controls.

Fiecare bandă din cadrul raportului va genera în timpul rulării lui, mai multe instanțe, în funcție de semnificația pe care o are. O bandă dintr-un raport reprezintă o formă generică a raportului final.



Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Conținutul benzilor descrise mai sus este următorul :

Page Header (Antet) se folosește pentru specificarea conținutului părții superioare a fiecărei pagini din raport. Ea va avea atâtdea instanțe câte pagini va avea raportul, și poate conține, de exemplu, numărul curent al paginii, data curentă, numele firmei care emite documentul, etc.

Detail va genera în raport rândurile de detaliu (conținutul de bază al raportului), adică câmpurile din tabela / tabele care se doresc să fie evidențiate în raport.

Page Footer (Subsol) va conține elementele necesare să fie afișate în subsolul paginii, cum ar fi, de exemplu, capitolul, ora listării, etc.

Fiecare bandă dintr-un raport are asociată o bară, care o delimită și cu ajutorul căreia poate fi redimensionată. Dimensiunea benzii corespunde dimensiunii zonei respective din pagina raportului final. Schimbarea dimensiunii se face prin tragerea, cu ajutorul mouse-ului, a barei delimitatoare asociate. Înălțimea minimă a unei benzii este zero, caz în care ea va fi lipită de banda anterioară.

Proprietățile unei benzii pot fi specificate de utilizator. Dând un dublu click pe bandă, se deschide fereastra de proprietăți în care putem specifica:

Height - înălțimea benzii. Dacă se dorește ca înălțimea să fie constantă pentru toate instanțele benzii, se activează comutatorul **Constant band height**

On Entry - expresia ce se verifică înainte de afișarea benzii

On exit - expresia ce se verifică după fiecare afișarea benzii

De exemplu, pentru menținerea unui contor intern care să memoreze numărul de pagini tipărite, se poate executa o instrucțiune de incrementare a controlului înainte sau după fiecare afișare a benzii de antet de pagină. Instrucțiunea va fi introdusă într-o funcție definită de utilizator, iar apelul ei va fi scris în una dintre expresiile **On Entry** sau **On Exit**.

Meniul Report conține următoarele opțiuni:

- **Title/Summary** – duce la afișarea unei ferestre de dialog ce permite, prin selecție, afișarea în cadrul raportului a două benzi suplimentare;
- **Title** (Titlu)- banda ce apare în raport o singura dată, la începutul acestuia, pe prima pagină, în care putem afișa de exemplu titlul listei și eventual capul de tabel, dacă acesta nu trebuie să fie repetat pe fiecare pagină ;
- **Summary** (Rezumat) – se afișează pe ultima pagină a raportului, o singură dată ;
- **Data Grouping**- duce la afișarea unei ferestre de dialog ce permite gruparea datelor din raport în funcție de anumite chei de grupare

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

ce sunt expresii formate din câmpuri ale tabelelor sau variabile definite. Pentru fiecare nou grup în raport apar două benzi și anume Group Header în (antetul grupului de nivel n) și Group Footer n (subsolul grupului de nivel n) ;

- **Variables** – se pot defini variabile necesare rulării raportului sau afișării unor anumite valori ;
- **Default Font** – stabilește fontul implicit cu care se lucrează în raport;
- **Private Data Session** - prin selectarea/deselectarea acestei opțiuni se indică dacă mediul de date folosit este privat sau nu;
- **Quick Report** – duce la afișarea unei ferestre din care putem selecta :
 - **Field Layout** - modul de dispunere a câmpurilor în cadrul raportului (orizontal sau vertical) ;
 - **Titles** – dacă dorim sau nu afișarea numelor de câmpuri;
 - **Add Alias** – dacă dorim sau nu prefixarea numelui câmpului cu cel al tablelei;
 - **Add table to data environment** – dacă dorim sau nu adăugarea tablelei la mediul de date;
 - **Run Report** – rularea raportului.

Bara **Report Controls** conține controalele necesare popularii benzilor raportului, și anume :

Select		selecție obiecte
Etichetă		folosit la introducerea textului
Câmp		afișarea câmpurilor din tablelele asociate
Elemente geometrice		pentru desenarea de linii, pătrate, cercuri, etc, folosite pentru cosmetizarea raportului
Imagine		necesar afișării de imagini grafice

Mediul de date al raportului

La construirea raportului este necesară stabilirea datelor sursă. Acestea, împreună cu anumite variabile folosite de raport, alcătuiesc mediul de date.

Mediul de date este un obiect, el are proprietăți și metode ce pot fi manipulate.

În mediul de date al unui raport sunt specificate tabelele sursă, ceea ce are ca efect deschiderea automată a acestora la pornirea raportului, fără a fi necesare instrucțiuni speciale în acest sens. Adăugarea unei tabele la mediul de date se face prin selectarea opțiunii **Add**. Dacă tabela este legată în cadrul unei baze de date, în mediul de date al raportului se vor încărca și legăturile respective.

În cazul rapoartelor care folosesc două sau mai multe tabele, este necesară specificarea în mediul de date și a legăturilor dintre ele, pentru ca informațiile să fie corect afișate.

Dacă se dorește specificarea dinamică a tabelelor sursă, se pot folosi metodele mediului de date, în care se vor introduce instrucțiunile corespunzătoare pentru deschiderea tabelelor.

Principalele metode ale mediului de date sunt :

- **Before Open Tables** – se stabilește numele și locația tabelelor;
- **Open Tables** – comenzi pentru deschiderea efectivă a tabelelor;
- **After Open tables** – se pot selecta tabela curentă și înregistrările curente;
- **Before Close Tables** – se pot efectua diverse calcule asupra datelor înainte de închiderea tabelelor;
- **Close Tables** – se închid efectiv tabelele;
- **After Close Tables** - codul ce se efectuează după închiderea tabelelor.

Principalele proprietăți ale mediului de date sunt :

- **InitialSelectedAlias** – desemnează tabela ce va fi deschisă inițial;
- **AutoOpenTables** – dacă este .T. determină deschiderea automată a tabelelor specificate;
- **AutoCloseTables** - dacă este .T. determină închiderea automată a tabelelor specificate.

Câmpurile unui raport (Fields)

Câmpurile reprezintă elementele variabile ale unui raport. Ele pot avea ca sursă:

- un câmp din tabelele folosite;

Facultatea de Automatică și Calculatoare Iași ***Baze de date – lucrări practice***

- o expresie formată din câmpurile tabelelor din mediul de date – caz în care expresia va fi evaluată de fiecare dată la rularea raportului pentru fiecare instanță a benzii respective;
- o variabilă a raportului.

Un câmp din raport se caracterizează prin :

- Sursa de date;
- Formatul de afișare (caracter, numeric, dată calendaristică, aliniere) ;
- Funcția de calcul (sumă, medie, numărare, etc) ;
- Condiții dinamice de afișare .

Câmpurile calculate dintr-un raport se obțin folosind funcții predefinite, disponibile la acționarea butonului **Calculation** a ferestrei **Report Expression**. Funcția se alege prin intermediu butonului de selecție.

Valoarea unui câmp calculat poate fi zerorizată la începerea unei pagini noi, începerea unui nou grup, terminarea unui grup, etc, momentul anularii fiind stabilit cu ajutorul listei derulante **Reset** a ferestrei **Calculated Fields**.

Gruparea datelor într-un raport

Este una dintre cele mai puternice facilități oferite de constructorul de rapoarte. Se pot construi rapoarte cu unul sau mai multe niveluri de grupare sau totalizare. Fiecare criteriu de grupare este caracterizat printr-o expresie ce va fi evaluată de fiecare instanță a benzii. Toate înregistrările ce corespund aceleiași valori a expresiei de grupare vor forma un grup, deci într-un raport cu niveluri de grupare vom avea atâtea grupuri câte valori are cheia de grupare. Dacă nu a fost specificată în prealabil ordonarea datelor după expresia (expresiile) de grupare, listarea informațiilor nu va fi concluzivă.

Specificarea expresiilor de grupare se face în fereastra **Data Grouping**.

Lista **Group Expressions** conține câte un rând pentru fiecare nivel de grupare, în care se poate specifica direct expresia de grupare. Dacă se dorește folosirea constructorului de expresii, se dă click pe butonul din dreapta câmpului de editare.

Pentru fiecare grup se pot specifica următoarele proprietăți:

- **Start group on new column** – grupul să înceapă în coloană nouă (în cazul unui raport multi - coloană). Dacă un grup se termină la mijlocul unei coloane, restul spațiului din coloană va fi lăsat liber și următorul grup începe în coloana următoare.
- **Start each group on a new page** – fiecare grup începe pe o pagină nouă.

- **Reset page number to 1 on each group** - pentru fiecare grup nou, numerotarea paginilor va începe de la 1.
- **Reprint group header on each page** - tipărește antetul grupului pe fiecare pagină, chiar dacă nu începe un nou grup, imediat după antetul de pagină.
- **Start group on new page when less than** -începe grupul pe pagină nouă când spațiul rămas pe pagina curentă este mai mic decât valoarea indicată de proiectantul raportului.

Rularea unui raport



Comanda folosită este :

REPO FORM <numeraport>

Pentru previzualizarea raportului

REPO FORM <numeraport> PREVIEW

Pentru trimitera listei direct la imprimantă:

REPO FORM <numeraport> TO PRINT

Pentru salvarea listei direct într-un fișier de tip.txt

REPO FORM <numefisier.txt> to <numefisier>

Dacă se dorește suprimarea afișării listei pe ecran se va adăuga clauza **Noconsole**. De obicei comanda de rulare a raportului se introduce în metoda **Click** a butonului numit de exemplu „Rulare Raport” din programul de raportare.

Programe de raportare

În general, același raport poate fi folosit pentru mai multe tipuri de liste, datele de raportare diferind în funcție de anumite variabile ce pot fi setate.

De exemplu, dacă se dorește raportare unei situații a vânzărilor de medicamente dintr-o farmacie, se pot specifica o serie de parametri cum ar fi : anul, luna, intervalul (data de început și cea de sfârșit), categoria de produse, etc. Etapele ce trebuie parcursă pentru realizarea raportului sunt următoarele:

- Se construiește o formă cu care se preiau de la utilizator variabilele necesare (cele specificate anterior). Forma va trebui să conțină în afară de câmpurile necesare preluării datelor și minim două butoane pentru rularea raportului și închiderea formei.
- Se efectuează preluarea datelor din tabele în funcție de parametrii specificați, adică se extrag datele, se grupează corespunzător într-o tabelă temporară, având grija ca numele câmpurilor să fie identice cu cele folosite în raport.
- Se dă clic pe butonul de executare a raportului și se afișează și verifică lista.

4.5 Constructorul de meniuri

Meniul reprezintă un ansamblu de opțiuni pus la dispoziția utilizatorului, opțiuni la alegerea cărora sunt declanșate diferite operații de prelucrare.

Meniul apare în partea superioară a ferestrei unei aplicații și are o structură standard, formată dintr-o bara orizontală (submeniu orizontal) ce conține mai multe opțiuni. Fiecare opțiune poate conține un submeniu vertical, care va fi activat numai la alegerea opțiunii respective.

Proiectarea unui meniu constă în specificarea elementelor componente (submeniuri și opțiuni) și a caracteristicilor acestora (legate de aspect și comportament).

Un meniu se construiește cu ajutorul constructorului de meniuri, care va genera un fișier de tip.**.mnx**. Apoi se va genera meniul, obținându-se un fișier de tip.**.mnt**. Obligatoriu, după orice modificare efectuată în meniu, acesta va trebui generat din nou, altfel modificarea efectuată nu este luată în considerație.

Comanda de creare a unui meniu este :

CREATE MENU <numemeniu>

sau alegând din meniul **File** opțiunea **New Meniu**, se deschide o fereastră din care se alege opțiunea :



- **Menu** dacă se dorește crearea unui meniu cu bară orizontală și submeniuri verticale;
- **Shortcut** dacă se dorește construirea unui submeniu vertical;

Modificarea unui meniu existent se face cu comanda :

MODIFY MENU <numemeniu>

sau alegând din meniul **File** opțiunea **Open**.

Proprietățile generale ale unui meniu se stabilesc în fereastra de dialog **General Options**, ce se deschide la alegerea opțiunii cu același nume a meniului **View**.

Posiția noului meniu relativ la meniul standard al sistemului este stabilită prin intermediu butoanelor din secțiunea **Location**. Opțiunile sunt :

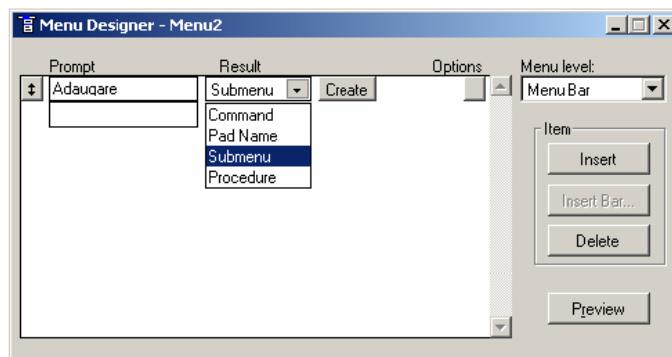
- **Replace** – Noul meniu îl înlocuiește pe cel al sistemului;
- **Append** – Noul meniu va fi adăugat la cel al sistemului;
- **Before** – Noul meniu va fi poziționat înaintea opțiunii din meniul sistem aleasă din lista derulantă;
- **After** – Noul meniu va fi poziționat după opțiunea din meniul sistem aleasă din lista derulantă.

Pot fi specificate sevențe de cod care să fie executate la apariția unor evenimente și care vor fi scrise în zona **Procedure** sau în fereastra deschisă prin apăsarea butonului **Edit**.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Se pot specifica secvențe de cod atât pentru inițializarea meniului **Setup** cât și pentru ștergerea sa **Cleanup**, în fereastra deschisă prin activarea comutatoarelor **Setup** sau **Cleanup** urmată de acționarea butonului OK.

Fereastra de lucru a constructorului de meniuri arată ca în figura de mai jos :



Prima coloană conține butonul de schimbare a poziției opțiunii - prin tragerea cu mouse-ul ea poate fi deplasată în sus sau jos.

În coloana **Prompt** se introduce textul explicativ al opțiunii, text ca va fi afișat ca bară de meniu (ex : Adăugare).

Coloana **Results** cuprinde o listă derulantă ce conține opțiunile descrise mai jos, fiecare dintre ele asociindu-se codul scris în coloana a patra, și anume :

- **Command** – la alegerea acestei bare de meniu se va executa comanda specificată în coloana patru (de exemplu rularea unei forme cu comanda: do form adăugare) ;
- **Pad Name** – Se definește numele padului ;
- **Submenu** – se va deschide un submenu care trebuie configurat după aceleasi reguli;
- **Procedure** – se scrie procedura ce trebuie rulată la accesarea acestei bare de meniu;
- **Coloana Options** duce la deschiderea unei ferestre în care se pot face anumite setări pentru respectiva bara de meniu, și anume :
 - **Key label** : combinația de taste, echivalentă cu click cu mouse-ul pe opțiunea respectivă, ce se determină prin apăsarea tastelor dorite;
 - **Key text** – textul suplimentar ce se va afișa lângă opțiune indicând combinația de taste aleasă;

Facultatea de Automatică și Calculatoare Iași **Baze de date – lucrări practice**

- **Skip for** – se scrie expresia ce va fi evaluată la fiecare accesare a opțiunii de meniu. Dacă rezultatul evaluării este adevărat, se permite executarea comenzi asociate, în caz contrar opțiunea nu va fi accesibilă. Aceasta facilitate este bine venită în cazul în care, de exemplu, într-o anumită secvență de meniu este permis accesul doar unor anumiți utilizatori, sau sunt dezactivate, la un moment dat, anumite secțiuni din meniu, în funcție de anumite condiții, pentru a preveni anumite erori de rulare a programului.
- **Message** – se indica textul explicativ asociat opțiunii de meniu.
- **Comment** – se pot introduce texte explicative, pentru o mai bună înțelegere a logicii programelor.

Previzualizarea meniului se poate face în orice moment, chiar dacă el nu a fost salvat, pentru verificarea codul scris și se realizează prin apăsarea butonului **PREVIEW** a submeniului **Menu**. În starea de previzualizare, meniul sistem este înlocuit cu cel în curs de editare. Ieșirea din previzualizare se face prin acționarea butonului OK.

Butoanele din secțiunea **Item** sunt folosite pentru adăugarea de noi elemente (Insert și Insert Bar) sau pentru ștergerea unor opțiuni (Delete).



Observații:

- scrierea caracterelor !< înaintea unora dintre caracterele din sirul din coloana Prompt determină ca acel caracter să fie folosit ca o tasta rapidă de accesare a opțiunii. Caracterul va apărea ca fiind subliniat.
- scrierea caracterului ! înaintea textului din coloana Prompt determină ca opțiunea respectivă să fie dezactivată, ea apărând pe ecran în culori ștersse.
- scrierea caracterelor !- în locul textului din coloana Prompt determină ca linia de meniu să fie considerată ca o bară delimitatoare între mai multe grupuri de opțiuni ale același submenu.

Orice meniu trebuie generat, operație ce se realizează prin alegerea opțiunii **Generate** a submeniului **Menu**. Se precizează numele meniului și calea în care acesta va fi salvat (salvarea se face într-un fișier cu extensia **.mpr**)

Comanda de rulare a unui meniu este **DO <numemeniu.mpr>** și ea este inclusă de obicei în programul principal al aplicației.

Revenirea la meniul sistem se face cu comanda
SET SYSMENU TO DEFAULT

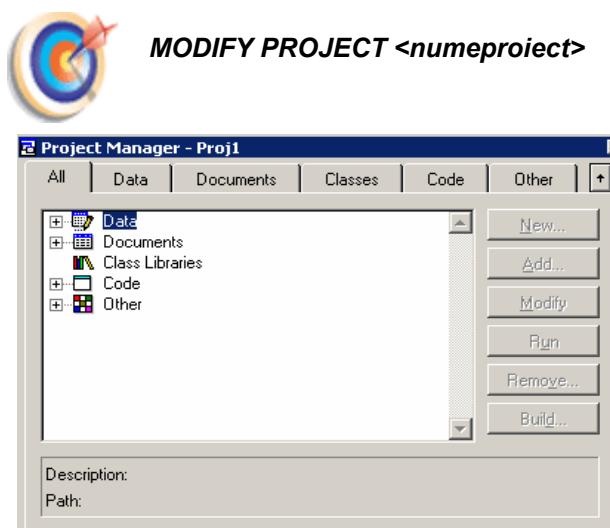
4.6 Constructorul de proiecte

Proiectele reprezintă înglobarea elementelor unui sistem informatic într-un tot unitar. Proiectul este o facilitate pusă la dispoziția proiectanților de sisteme informatiche, cu ajutorul căreia se poate ține cu ușurință evidența tuturor componentelor unui sistem, cum ar fi baze de date, tabele, vederi, programe, forme, meniuri, rapoarte, interogări, etc.

Se recomandă folosirea proiectelor în momentul în care sistemul informatic ce urmează a fi proiectat va conține un număr mare de elemente. În acest caz se va crea mai întâi proiectul, iar în carul lui se vor defini, pe rând, toate elementele necesare. Toate operațiile ce se efectuează asupra unui element din cadrul proiectului se vor face din interiorul gestionarului de proiecte astfel încât proiectul să fie în permanenta actualizat.

Proiectul este un fișier cu extensia **.pjx**.

Crearea unui proiect se face cu comanda:



Dacă proiectul există, acesta va fi deschis, dacă nu se va crea sau din meniul **File**, opțiunea **New Project** și se actionează butonul **New File**. Tipurile de elemente ce pot fi incluse într-un proiect sunt: baze de date, tabele libere, interogări, forme, rapoarte, etichete, biblioteci de clase, programe, biblioteci API,

aplicații, meniuri, fișiere text, alte tipuri de fișiere.

Fereastra ce se deschide la pornirea generatorului de proiecte arată ca în figura de mai sus și conține mai multe pagini corespunzătoare grupării logice a elementelor componente.

Adăugarea de noi elemente la un proiect se face astfel: se activează pagina proiectului în care va fi inclus respectivul element și se selectează din

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

listă tipul elementului de adăugat. Dacă elementul a fost deja creat în afara proiectului și se dorește adăugarea lui la proiect, se apasă butonul **Add**. Dacă elementul urmează să fi creat se apasă butonul **New**.

Editarea unui element din proiect se face prin selectarea elementului urmată de apăsarea butonului **Modify**.

Rularea unui element din proiect se face prin selectarea elementului urmată de apăsarea butonului **Run**.

Înlăturarea unui element din proiect se face prin selectarea elementului urmată de apăsarea butonului **Remove**.

Recompilarea elementelor proiectului se face prin apăsarea butonului **Build**. În acest caz poate fi aleasă una dintre opțiunile specificate în secțiunea **Options** a ferestrei **Build Options**, și anume :

- **Recompile all files** – recompilarea tuturor fișierelor;
- **Display errors** – afișarea erorilor;
- **Run after build** – rulare după recompilare.

În fereastra **Build Options** există și secțiunea **Action** ale cărei opțiuni sunt următoarele :

- **Rebuild project** – recompilarea proiectului;
- **Build Application** – se realizează o aplicație. Aplicația este un tip special de fișier ce conține toate elementele unui sistem informatic și care poate fi rulată în mediu VisualFox. Elementele componente ale unei aplicații pot fi împărțite în două mari categorii și anume: elementele ce fac parte integrantă din proiect, cum ar fi meniuri, forme, rapoarte, care nu se modifică la rulare, și elemente ce nu fac parte integrantă din aplicație (baze de date, tablele) ale căror conținut se actualizează permanent în momentul rulării. Orice aplicație se construiește în jurul unui program principal (acel program care va fi lansat primul la rularea aplicației). Stabilirea lui se face prin selectarea lui și alegerea opțiunii **Set Main** (click buton dreapta din meniul rapid).
- **Build Executable** – VisualFoxPro permite rularea programelor în varianta compilată și a aplicațiilor. Programele de acest tip nu pot rula în afara mediului fox și de aceea ele trebuie construite într-o formă executabilă. Pentru aceasta trebuie să îndeplinească condițiile expuse la generarea aplicațiilor și alegerea butonului **Build executable**. Va fi generat un fișier cu extensia .exe

Rularea unei aplicații se face cu comanda:

DO <nume_aplicatie.app>

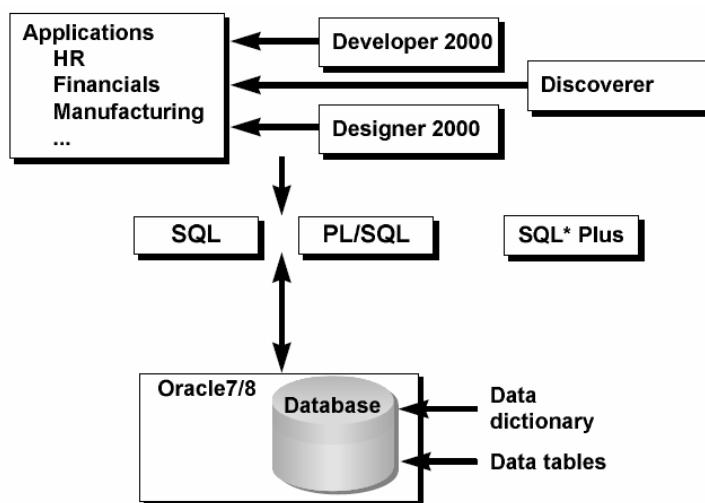


4.7 Exerciții:

1. Construiți un raport cu ajutorul căruia să puteți lista agenții în ordine alfabetică. (folosiți fișierul Ag.dbf și afișați câmpurile: nume, prenume, data nașterii, data angajării, salariul de bază).
2. Construiți un raport pentru a lista agenții ce au efectuat tranzacții într-o perioadă specificată, în ordinea datei la care a fost efectuata tranzacția (folosiți fișierele Ag.dbf, Tr.dbf și afișați câmpurile cod agent, nume și prenume agent, data tranzacției).
3. Construiți un raport pentru a lista tranzacțiile efectuate într-o anumită luna dintr-un anumit an, specificând totalul valoric pentru operațiile de vânzare și cumpărare pentru fiecare zi în parte și deasemeni un total general la sfârșit de luna (folosiți fișierele Cl.dbf, Tr.dbf Ma.dbf și afișați câmpurile nume client, data tranzacției, valoarea de intrare, valoare de ieșire).
4. Construiți o formă care să conțină o casetă de tip listă care să permită selectarea unui anumit autor din tabela de autori, pentru care să listați, în ordine alfabetică, toate cărțile existente în bibliotecă, într-un fișier de tip.txt, pe care să-l afișați apoi pe ecran.
5. Construiți un raport care să va permite afișarea imaginilor grafice. De exemplu, puteți lista toate mărfurile din fișierul Ma.dbf, grupând datele în funcție de codul produsului, iar la fiecare cod produs puteți asocia elemente grafice diferite, reprezentative.
6. Construiți un meniu în care să folosiți toate opțiunile puse la dispoziție de constructorul de meniuri și în care să includeți rularea unor forme, rapoarte și programe construite în laboratoarele anterioare.

Capitolul 5 Limbajul SQL implementat în Oracle

5.1 Soluția completă Oracle



Oracle SGBDR (Sistem de Gestire Baze de Date Relaționale) este produsul de bază al Oracle. El include Oracle Server și mai multe instrumente pentru a ajuta utilizatorii în menținere, monitorizarea și utilizarea datelor. Dicționarul de date Oracle este o componentă foarte importantă a Serverului. El conține un set de tabele și view-uri care prezintă o imagine read-only a bazei de date.

SGBDR gestionează sarcini ca:

- managementul stocării și definirii datelor;
- controlul și restricționarea accesului la date și concurență;
- posibilități de salvare și restaurare;
- interpretează instrucțiuni SQL și PL/SQL.

Instrucțiunile SQL și PL/SQL sunt folosite de programe și utilizatori pentru accesul și manipularea datelor în baze de date Oracle. SQL*Plus este un instrument Oracle care recunoaște și trimite către server sintaxe SQL și PL/SQL spre execuție și conține propriile comenzi.

Oracle furnizează o mare varietate de instrumente GUI pentru construirea aplicațiilor, precum și o gamă largă de aplicații software pentru afaceri și industrie.

SQL și SQL*Plus

SQL este un limbaj de comenzi pentru comunicarea cu Serverul Oracle din orice instrument sau aplicație. SQL Oracle conține multe extensii. Când se introduce o instrucțiune SQL, aceasta este stocată într-o zonă de memorie numită buffer SQL și este disponibilă până la introducerea unei noi instrucțiuni.

SQL*Plus este un instrument Oracle ce conține propriu limbaj de comenzi și care recunoaște și trimit instrucțiuni SQL la server pentru execuție.

Caracteristicile limbajului SQL

- Poate fi folosit de o gamă largă de utilizatori, inclusiv de cei care nu sunt programatori;
- Este un limbaj neprocedural;
- Reduce timpul necesar pentru crearea și menținerea sistemelor;
- Sintaxa limbajului este în limba engleză.

Caracteristicile limbajului SQL*Plus

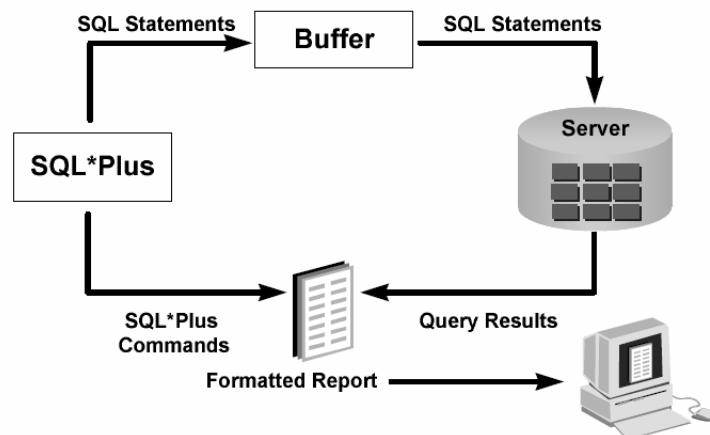
- Acceptă fișiere cu instrucțiuni SQL;
- Furnizează un editor în mod linie pentru modificarea instrucțiunilor SQL;
- Controlează setările de mediu;
- Formatează rezultatele interogărilor în rapoarte;

Comparatie între SQL și SQL*Plus

SQL	SQL*Plus
Este un limbaj pentru comunicarea cu serverul Oracle pentru accesarea datelor	Recunoaște instrucțiuni SQL și le trimit la server
Este bazat pe standardul ANSI SQL	Este o interfață proprietate Oracle pentru executarea instrucțiunilor SQL
Manipulează date și definiții de tabele în baze de date	Nu permite manipularea valorilor în baze de date
O instrucțiune se stochează în bufferul SQL, pe una sau mai multe linii	Este permisă o singură comandă pe linie care nu este stocată în bufferul SQL
Nu are caracter de continuare	Are caracterul (-) pentru continuarea unei comenzi dacă aceasta nu începe pe o singură linie
Folosește un caracter de terminare pentru executarea imediată a comenzi	Nu are nevoie de caractere de terminare. Se executa imediat
Nu pot fi abreviate	Pot fi abreviate
Folosește funcții pentru anumite formatari	Folosește comenzi pentru formatari

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Interacțiunea SQL și SQL*Plus



SQL*Plus este un mediu care permite:

- Conectarea la SQL*Plus;
- Afisarea structurii unei tabele;
- Editarea instructiunilor SQL;
- Executarea instructiunilor SQL din SQL*Plus;
- Salvarea instructiunilor SQL in fisiere;
- Executarea fisierelor salvate;
- Încărcarea comenzi din fisier in buffer pentru editare;
- Executarea instructiunilor SQL pentru a extrage, modifica, adauga și șterge date din baza de date;
- Formatarea, calcularea, stocarea și listarea rezultatelor interogărilor sub forma de rapoarte;
- Crearea fisierelor de script pentru stocarea instructiunilor SQL.

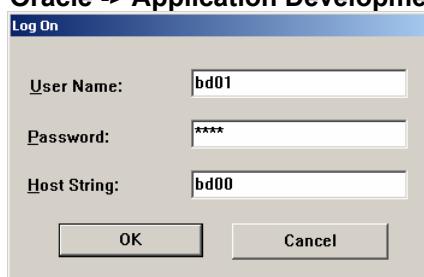
Comenzi SQL*Plus

Categorie	Scop
Mediu	Afectează comportamentul general al instructiunilor SQL pentru sesiunea respectivă
Formatare	Formatează rezultatele interogării
Manipulări de fisiere	Salvează, încarcă și rulează fisiere de script
Execuție	Trimite instructiuni SQL din bufferul SQL la serverul Oracle
Editare	Modifică instructiuni SQL în buffer
Interacțiune	Permite crearea și trimiterea variabilelor la instructiuni SQL, afișarea variabilelor și listarea mesajelor pe ecran
Diverse	Are diferite comenzi pentru conectarea la baza de date, manipularea mediului SQL*Plus și afișarea coloanelor

Conecțarea la SQL*Plus

SQL*Plus se poate apela în funcție de tipul sistemului de operare sau a mediului Windows în care se rulează. Pentru conectarea într-un mediu Windows:

Start -> Programs -> Oracle for Windows NT -> SQL*Plus sau
Start -> Programs -> Oracle -> Application Development -> SQL*Plus



Se completează: username, parola și baza de date

Pentru conectarea într-un mediu de tip linie de comandă se lansează următoarea comandă:

Sqlplus [username[/password[@database]]]

unde username = numele utilizatorului din baza de date

 password = parola de conectare la baza de date

 @database = sirul de conectare la baza de date



NOTĂ: Pentru a nu deconspira parola, se introduce numele utilizatorului și apoi la prompterul Password se introduce parola și sirul de conectare.

5.2 Comenzi SQL*Plus pentru fișiere

SAVE filename	salvează conținutul buferului SQL într-un fișier.
GET filename	scrie conținutul unui fișier în buffer SQL (extensia predefinită este.sql)
START filename	rulează un fișier script
@ filename	la fel ca START
ED[IT]	lansează editorul și salvează conținutul bufferului într-un fișier afiedt.buf
ED[IT] filename	lansează editorul pentru editarea conținutului unui fișier salvat

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

SPO[OL] [filename]OFF OUT	stochează rezultatul unei interogări într-un fișier. OFF închide fișierul OUT închide fișierul și îl trimite la imprimanta sistem
EXIT	părăsește mediul SQL*Plus

5.3 Instrucțiuni SQL

Extragere de date

SELECT

Limbajul de manipulare a datelor (DML – Data Manipulation Language) (introduce rânduri noi, le șterge pe cele nedorite și le actualizează pe cele existente deja în tabele)

INSERT

UPDATE

DELETE

Limbajul de definire a datelor (DDL- Data Definition Language) (setează, schimbă sau șterge structuri de date din tabele)

CREATE

ALTER

DROP

RENAME

TRUNCATE

Controlul tranzacțiilor (gestionează schimbările făcute de instrucțiunile DML; actualizările efectuate asupra datelor pot fi grupate împreună în tranzacții logice)

COMMIT

ROLLBACK

SAVEPOINT

Limbajul de control al datelor (DCL- Data Control Language) (acordă sau retrage drepturi de acces asupra bazelor de date Oracle și a utilizatorilor săi)

GRANT

REVOKE

Oracle SQL este compatibil cu standardele acceptate de industrie. Comitetele de standardizare acceptate de industrie sunt ANSI (Institutul American pentru Standarde) și ISO (Organizația Internațională pentru Standarde). Atât ANSI cat și ISO au acceptat SQL ca limbajul standard pentru baze de date relaționale.

Tabelele folosite în acestă carte sunt:

- **EMPLOYEES**, tabela care conține detalii despre angajați;
- **DEPARTMENTS**, tabela care conține detalii despre departamente;
- **JOB_GRADES**, tabela care conține detalii despre salarii și trepte de salarizare;

5.4 Sintaxa de bază a instrucțiunilor SQL

Comanda **SELECT** extrage informații din bazele de date. Folosind comanda **SELECT**, se pot face următoarele :

- **SELECTIE (SELECTION)**: poate fi folosită pentru a alege liniile necesare din tabelele de date. Se pot folosi criterii diferite de selecție.
- **PROIECTARE (PROJECTION)**: poate fi folosită pentru a alege coloanele din tabele necesare în interogarea rezultat. Se pot alege oricâte coloane de tabele.
- **COMBINAREA (JOIN)**: poate fi folosită pentru a uni datele aflate în tabele diferite prin crearea unei legături între coloanele tabelelor de unde provin datele.

Select - Sintaxa de baza



SELECT [DISTINCT] {*, column [alias],}
FROM table ;

SELECT pentru identificarea coloanelor
FROM pentru identificarea tabelelor

Într-o formă simplă, instrucțiunea SELECT include următoarele clauze :

- **SELECT**, care specifică ce coloane vor fi afișate;
- **FROM**, care specifică tabelele ce conțin coloanele scrise în clauza **SELECT**.

Din punct de vedere sintactic:

SELECT este o listă de una sau mai multe coloane;
DISTINCT suprimă duplicatele;
***** selectează toate coloanele;
column numele coloanei/coloanelor;
alias dă coloanei selectate un alt nume;
FROM specifică tabela/tabelele care conține/conțin coloanele.



NOTĂ : În această carte, cuvintele : “cuvânt cheie”, “clauza” , “instructiune” vor fi folosite astfel:

- Un “cuvânt cheie” se referă la un element SQL individual. De exemplu, SELECT și FROM sunt cuvinte cheie.
- O “clauză” este o parte dintr-o instructiune SQL.. De exemplu, SELECT employee_id, first_name,.. reprezintă o clauză.
- O “instructiune” este o combinație de două sau mai multe clauze și cuvinte cheie. De exemplu, SELECT * FROM employee este o instructiune SQL.

Scrierea instructiunilor SQL

Utilizând următoarele reguli se pot construi instructiuni valide, ușor de citit și de editat:

- Instructiunile SQL pot fi scrise cu litere mari sau mici, în afară de cazurile indicate;
- Instructiunile SQL pot fi introduse pe una sau mai multe linii;
- Cuvintele cheie nu pot fi abreviate sau despărțite pe linii diferite;
- De obicei clauzele sunt plasate pe linii separate pentru a fi lizibile;
- De obicei cuvintele cheie sunt introduse cu majuscule. Toate celelalte cuvinte, cum ar fi numele de tabele și coloane sunt introduse cu litere mici. Aceasta este doar o convenție de NOTĂre, nu o regula.
- În cadrul SQL*Plus, instructiunile SQL sunt introduse de la prompterul SQL, iar următoarele linii sunt numerotate. Acesta este un buffer SQL. Doar o singura instructiune poate fi adusă la un moment dat din buffer.



Executarea instructiunilor SQL se face urmând regulile:

- Poziționarea caracterului punct și virgulă (;) la sfârșitul ultimei clauze;
- Poziționarea unui slash (/) la sfârșitul ultimei linii din buffer;
- Scrierea unui slash la prompterul SQL;
- Comanda RUN sau @ la prompterul SQL.

Exemplu : pentru selectarea tuturor coloanelor și liniilor dintr-un tabel scriem

select * from departments;

Se pot afișa toate coloanele din tabela folosind cuvântul cheie **SELECT** urmat de un asterix (*). În exemplu, tabela departments conține coloanele: **DEPARTMENT_ID**, **DEPARTMENT_NAME**, **MANAGER_ID**, **LOCATION_ID**. Tabelul conține linii pentru fiecare departament.

Se pot afișa toate coloanele din tabela scriind toate coloanele după cuvântul cheie SELECT. Instrucțiunea SQL de mai sus afișează toate coloanele și toate liniile din tabela DEPARTMENTS.

5.5 Crearea și gestionarea tabelelor

Obiective:

- Descrierea obiectelor din baza de date;
- Crearea tabelelor;
- Descrierea tipurilor de date ce pot fi utilizate în momentul definirii specificațiilor pentru coloane;
- Modificarea structurii unei tabele;
- Ștergerea, redenumirea și trunchierea tabelelor.

Obiectele bazei de date

Obiect	Descriere
Table	Unitatea de baza pentru stocare compusă din lini și coloane.
View	Reprezentare logică a unor date dintr-o sau mai multe tabele.
Sequence	Generează valori pentru chei primare.
Index	Mărește viteza în cazul interogărilor.
Synonym	Nume alternative date obiectelor.

O baza de date Oracle poate conține structuri de date multiple. Fiecare structură trebuie definită la proiectarea bazei de date, astfel încât să poată fi creată în momentul construirii bazei de date.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Structura tabelelor in Oracle9i

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
<hr/>			
10	<i>Administration</i>	200	1700
20	<i>Marketing</i>	201	1800
30	<i>Purchasing</i>	114	1700
40	<i>Human Resources</i>	203	2400
50	<i>Shipping</i>	121	1500
60	<i>IT</i>	103	1400
70	<i>Public Relations</i>	204	2700
80	<i>Sales</i>	145	2500
90	<i>Executive</i>	100	1700
100	<i>Finance</i>	108	1700
110	<i>Accounting</i>	205	1700

Tabelele pot fi create în orice moment, chiar în momentul când utilizatorul folosește baza de date. Nu este obligatorie specificarea mărimei unei tabele. Este important de știut ce spațiu va ocupa tabela după un timp. Structura unei tabele poate fi modificată dinamic.

Convenții pentru denumirea tabelelor

- Numele trebuie să înceapă cu o literă;
- Numele poate avea lungimea de 1-30 caractere;
- Caracterele permise sunt numai A-Z, a-z, 0-9, \$ și #;
- Un utilizator nu poate avea în schema sa două tabele cu nume identice;
- Numele tabelei nu poate fi un nume rezervat (de exemplu: select, create, define).
- Se folosesc nume descriptive pentru tabele sau alte obiecte din baza de date;
- Se folosesc aceleași nume pentru aceleași entități din tabele diferite. De exemplu, coloana cu numărul departamentului este denumita DEPTNO atât în tabela EMPLOYEES cât și în tabela DEPARTMENTS.

5.6 Comanda CREATE TABLE

Pentru a crea o tabelă utilizatorul trebuie să dețină :

- Drepturi pentru crearea unei tabele (drepturi Oracle);
- Spațiu de stocare în Oracle (drepturi Oracle).

La crearea unei tabele trebuie specificate:

- Numele tabelei;
- Numele coloanei, tipul de dată al coloanei și dimensiunea maximă a acesteia.



CREATE TABLE [schema.] table (column datatype [DEFAULT expr]);

Sintaxa:

- schema numele posesorului tabelei;
- table numele tabelei;
- DEFAULT specifică valoarea implicită ;
- column numele coloanei;
- datatype tipul de dată și lungimea.

Referirea tabelelor aflate în schema unui alt utilizator

Schema este o colecție de obiecte. Obiectele din schemă sunt structuri logice care se referă direct la datele din baza de date. Obiectele din schemă includ tabele, view-uri, sinonime, secvențe, proceduri stocate, indecsi, clustere și legăturile bazei.

Tabelele a căror proprietar este un alt utilizator nu sunt în schema utilizatorului curent. Pentru a fi referite trebuie folosit numele proprietarului tabelei. Aceasta trebuie scris înaintea numelui tabelei, urmat de punct. (exemplu: bd03.employees pentru tabela employees a utilizatorului bd03).

Opțiunea DEFAULT

Specifică valoarea implicită pentru o coloană, într-o operație de inserare.

Select hiredate DATE DEFAULT SYSDATE from employees;

- Valorile permise sunt valori literale, expresii sau funcții SQL;
- Valorile ilegale sunt numele altor coloane sau pseudocoloane;
- Valoarea implicită trebuie să aibă același tip cu cel al coloanei.

Unei coloane i se poate asigna o valoare implicită utilizând opțiunea **DEFAULT**. Această opțiune previne atribuirea unor valori de **null** pentru datele inserate fără precizarea unei valori explicate. Valoarea poate fi un literal, o expresie sau o funcție SQL, cum ar fi **SYSDATE** sau **USER**, dar valoarea nu poate fi cea a unei alte coloane sau o pseudocoloane, cum ar fi **NEXVAL** sau **CURRVAL**. Valoarea implicită trebuie să fie de același tip cu tipul de data al coloanei.



Exemplu : crearea tabeli departments

```
SQL> create table departments
2 (department_id number(4),
3 department_name varchar2(30),
4 manager_id number(6),
5 location_id number(4));
```

Table created.

Confirmarea creării tabeli:

```
SQL> desc departments;
Name Null? Type
-----
DEPARTMENT_ID NUMBER(4)
DEPARTMENT_NAME VARCHAR2(30)
MANAGER_ID NUMBER(6)
LOCATION_ID NUMBER(4)
```

Exemplul anterior creează tabela Departments, cu coloanele: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID. Mai departe se confirmă crearea tabelei DEPT ca rezultat al comenzi **DESCRIBE**.

Deoarece instrucțiunea **CREATE TABLE** este una de tip DML, la rularea ei se execută în mod automat și o instrucțiune de tip **COMMIT**.

5.7 Tabele din baza de date Oracle

În baza de date Oracle există două tipuri de tabele și anume:

- Tabele utilizator ce reprezintă o colecție de tabele create și administrate de utilizator ce conțin informațiile utilizatorilor;
- Dictionarul de date ce reprezintă o colecție de tabele create și administrate de Oracle Server ce conțin informații despre baza de date.

Toate tabelele din dicționarul de date sunt proprietatea utilizatorului SYS și sunt accesate rar de către utilizatori pentru că informațiile pe care acestea le conțin sunt greu de înțeles. De aceea, în mod obișnuit, utilizatorii accesează vederile din dicționarul de date pentru că informațiile sunt prezentate într-o formă ușor de înțeles.

Informațiile stocate în dicționarul de date conțin:

- numele utilizatorilor bazei Oracle și drepturile acestora ;
- obiectele bazei de date – numele lor, constrângerile și informații legate de audit.

În dicționarul de date există patru categorii de view-uri, fiecare dintre ele având un prefix distinct ce reflectă scopul în care poate fi folosit.

- USER_View - conțin informații despre obiectele aflate în proprietatea userului.
- ALL_View - conțin informații despre toate tipurile de tabel (obiecte tabel și tabele relaționale) accesibile utilizatorului.
- DBA_View - Aceste view-uri sunt restricționate, ele putând fi accesate doar de cei care au rolul de DataBase Administrator.
- V\$ - Aceste view-uri sunt view-uri cu performanțe dinamice, performanțe ale serverului de baze de date, memorie și blocări.

5.8 Interogarea dicționarului de date



Afișarea numelor tabelelor (coloana table_name) unui utilizator, precum și alte informații legate de aceste tabele:

SQL> select * from user_tables;

Afișarea tipurilor de obiecte ale unui utilizator:

SQL> select distinct object_type from user_objects;

Afișarea tabelelor, view-urilor, sinonimelor și secvențelor unui utilizator:

SQL> select * from user_catalog;

Puteți interoga dicționarul de date pentru a afișa informații diverse despre obiectele de tip baza de date ale unui utilizator. Tabelele cele mai des utilizate din dicționarul de date sunt **USER_TABLES**, **USER_OBJECTS** și **USER_CATALOG**. Tabela **USER_CATALOG** are un sinonim numit **CAT**. Puteți folosi acest nume în loc de **USER_CATALOG** în comenzi.

SQL> select * from cat;

5.9 Tipuri de date

Tipul de data	Descriere
VARCHAR2(size)	Dată de tip caracter, de lungime variabilă. (Trebuie specificată o lungime maximă - size, valoarea implicită este 1,maxima 4000)
CHAR(size)	Dată de tip caracter de lungime fixă. (Valoarea implicită este cea minimă 1, maxima 2000)
NUMBER(p,s)	Număr cu p cifre și s cifre zecimale. (p poate lua valori între 1 și 38, s între 84 și 127)
DATE	Dată de tip dată calendaristică cuprinsă între Ianuarie, 1, 4712 B.C. și Decembrie, 31, 9999 A.D.
LONG	Dată de tip caracter cu lungime variabilă, până la 2 gigabytes.
CLOB	Dată de până la 4 gigabytes.
RAW(size)	Dată binară cu lungime specificată. Valoarea maximă este de 2000. Aceasta valoare trebuie specificată.
LONG RAW	Dată binară cu lungime de până la 2 gigabytes.
BLOB	Dată binară cu lungime de până la 4 gigabytes.
BFILE	Dată binară stocată într-un fișier extern, lungime maximă de până la 4 gigabytes.



NOTĂ

- coloana de tip LONG nu este copiată atunci când la crearea tabelei se folosește o subinterrogare;
- coloana de tip LONG nu poate fi inclusă într-o clauza ORDER BY sau GROUP BY;
- într-o tabela poate să existe doar o coloană de tip LONG;
- Nu se pot defini constrângeri asupra unei coloane de tip LONG.

Oracle9i aduce îmbunătățiri asupra modului de stocare a datei și timpului prin introducerea de noi tipuri de date pentru dată calendaristică și timp și corelarea cu zonele geografice **TIMESTAMP**.

TIMESTAMP stochează anul, luna și ziua din tipul de dată DATE plus valorile pentru oră, minut și secundă precum și fracțiuni de secundă și reprezintă o extensie a tipului de dată **DATE**. Specificarea acestui tip de data se face cu:



TIMESTAMP[(fractional_seconds_precision)]

unde **(fractional_seconds_precision)** poate lua valori între 0 și 9, implicit fiind 6

Exemplu



```
CREATE TABLE new_employees  
(employee_id NUMBER,  
first_name VARCHAR2(15),  
last_name VARCHAR2(15), ...  
start_date TIMESTAMP(7), ...);
```

În exemplu de mai sus se creează tabela NEW_EMPLOYEES ce are coloana start_date de tip TIMESTAMP. Valoarea '7' indică precizia fracțiunilor pentru secunde. Presupunem că s-au introdus două rânduri în tabela NEW_EMPLOYEES.

Ieșirile arată diferența de afișare dintre o valoare de tip de dată DATE, care este afișată în formatul DD-MON-YY și una de tip TIMESTAMP.

```
SELECT start_date FROM new_employees;
```

```
17-JUN-87 12.00.00.0000000 AM  
21-SEP-89 12.00.00.0000000 AM
```

TIMESTAMP WITH TIME ZONE este o variantă a formei **TIMESTAMP** care include și afișarea timpului pentru zona geografică.

Afișarea cu TIME ZONE reprezintă diferența în ore minute și secunde dintre timpul local și UTC (Coordinated Universal Time, adică Greenwich Mean Time).

TIMESTAMP[(fractional_seconds_precision)] WITH TIME ZONE

Două valori de tip TIMESTAMP WITH TIME ZONE sunt considerate identice dacă ele reprezintă același moment în UTC din punct de vedere al decalării în timp a valorilor TIME ZONE stocate.

De exemplu :

TIMESTAMP '1999-04-15 8:00:00 -8:00' este identic cu
TIMESTAMP '1999-04-15 11:00:00 -5:00'

adică 8:00 a.m. Pacific Standard Time este identic cu 11:00 a.m. Eastern Standard Time.

Acest lucru poate fi specificat ca :

TIMESTAMP '1999-04-15 8:00:00 US/Pacific'



TIMESTAMP WITH LOCAL TIME ZONE este o altă variantă pentru TIMESTAMP care include afișarea valorii timpului local.

Data stocată în baza este normalizată într-o baza ce conține timpul local. Timpul local nu este stocat ca parte a unei coloane de tip DATE. Atunci când se inițiază o sesiune locală serverul returnează data în formatul ce afișează timpul local.

TIMESTAMP WITH LOCAL TIME ZONE are următoarea sintaxă:

TIMESTAMP[(fractional_seconds_precision)] WITH LOCAL TIME ZONE

Spre deosebire de TIMESTAMP WITH TIME ZONE, puteți specifica ca o coloană de tipul TIMESTAMP WITH LOCAL TIME ZONE ca fiind parte a unei chei primare sau unice.

Afișarea timpului va preciza diferența (în ore și minute) dintre timpul local și timpul UTC.



Exemplu

```
CREATE TABLE time_example AS (order_date TIMESTAMP  
WITH LOCAL TIME ZONE);
```

```
INSERT INTO time_example VALUES('15-NOV-00 09:34:28 AM');  
SELECT * FROM time_example;
```

order_date

```
-----  
15-NOV-00 09.34.28 AM
```



INTERVAL YEAR TO MONTH stochează o perioadă de timp folosind câmpurile an și luna din data. Sintaxa este :

INTERVAL YEAR [(year_precision)] TO MONTH

unde *year_precision* este numărul de cifre în care se va afișa anul, implicit 2.



Restricție

Prima valoare trebuie să fie semnificativ mai mare decât cea de-a doua. De exemplu: INTERVAL '0-1' MONTH TO YEAR nu este validă.

Exemplu:



INTERVAL '312' YEAR(3) indică un interval de 312 ani și 0 luni.
INTERVAL '300' MONTH(3) indică un interval de 300 luni.
INTERVAL '312-2' YEAR(3) TO MONTH indică 312 ani și 2 luni.

Crearea unei tabele utilizând o interogare

Creați o tabelă și inserăți linii combinând comanda **CREATE TABLE** cu opțiunea **AS subquery**;



```
CREATE TABLE table  
[column (,column...)]  
AS subquery ;
```

Numărul coloanelor tabelei trebuie să fie egal cu numărul coloanelor din subinterrogare;

Utilizând opțiunea **AS subquery** se creează tabela și inserează înregistrările furnizate de interogare.

Sintaxa:

- | | |
|-------------------|---|
| - <i>table</i> | numele tabelei; |
| - <i>column</i> | numele coloanei, valoarea implicită și constrângerile de tip; |
| - <i>subquery</i> | comanda SELECT care definește un set de înregistrări ce trebuie inserate în tabela. |

- Coloanele tabelei au numele specificate iar rândurile ce sunt inserate sunt cele rezultate din fraza SELECT din subinterrogare.
- La definirea coloanelor se poate specifica doar numele coloanei și valoarea implicită.
- Dacă se dorește specificarea coloanelor, numărul acestora trebuie să fie egal cu numărul de coloane rezultat din subinterrogare.
- Dacă nu se specifică coloanele, numele coloanelor sunt identice cu cele date de subinterrogare.
- Regulile de integritate nu sunt preluate de noua tabelă.



Exemplu:

```
CREATE TABLE dept80
AS
SELECT employee_id, last_name,
       salary*12 ANNSAL,
       hire_date
FROM employees
WHERE department_id = 80;
```

Table created.

DESCRIBE dept80

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

În exemplu se creează tabela DEPT80 ce conține date despre angajații ce lucrează în departamentul 80. Observați că datele au fost aduse din tabela EMPLOYEES.

Puteți verifica existența tabelei și a definiției coloanelor folosind comanda iSQL*Plus DESCRIBE.



Verificați dacă ați redenumit coloanele ce conțin expresii.
În exemplul nostru coloana SALARY*12 a fost redenumita ANNSAL.
Fără acest alias s-ar fi generat mesajul de eroare
ERROR at line 3:
ORA-00998: must name this expression with a column alias

5.10 Comanda ALTER TABLE

Comanda este utilă dacă se dorește modificarea structurii unei tabele. Se pot adăuga noi coloane utilizând clauza **ADD**, modifica coloane existente sau defini valori implicate pentru coloane utilizând clauza **MODIFY**, șterge coloane folosind clauza **DROP**.

Adăugarea unei coloane



```
ALTER TABLE table
ADD (column datatype [DEFAULT expr]
     [, column datatype....]);
ALTER TABLE table
MODIFY (column datatype [DEFAULT expr]
     [, column datatype....]);
ALTER TABLE table
DROP COLUMN (column);
```

Sintaxa:

- *table* numele tablei;
- *column* numele noii coloane;
- *datatype* tipul datei și lungimea;
- *DEFAULT expr* specifică valoarea implicită pentru coloană.



EXEMPLU:

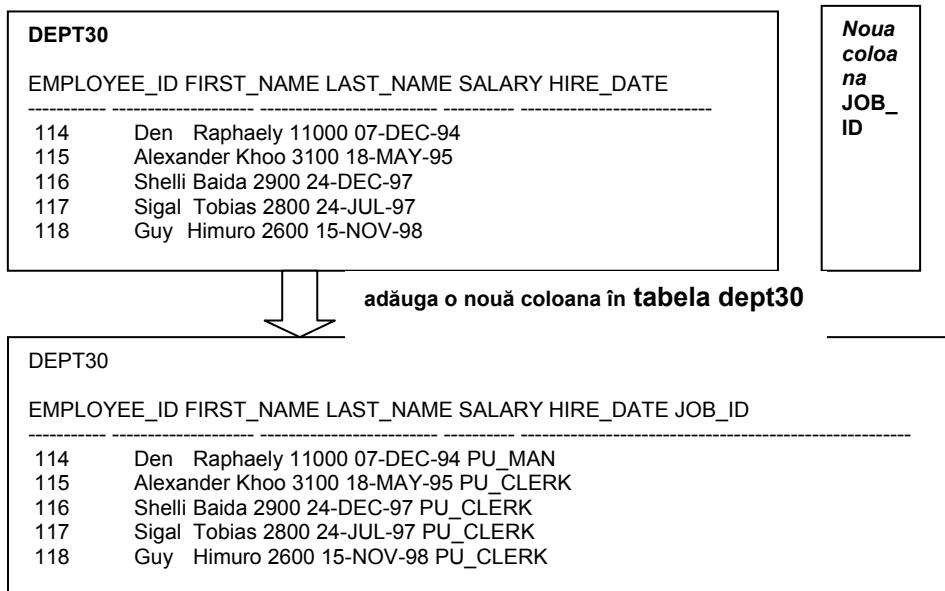
```
ALTER TABLE dept30 ADD (job_id varchar2(10));
Table altered.
```



Observații:

- Se pot adăuga, modifica sau șterge coloane din tabela. Facilitatea de a șterge o coloana dintr-o tabela a apărut în versiunea Oracle 9i.
- Nu puteți specifica locul de apariție al noii coloane. Noua coloană devine automat ultima coloană. Dacă tabela conține înregistrări în momentul adăugării unei noi coloane, atunci noua coloană se initializează cu valori nule pentru toate înregistrările.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



Modificarea unei coloane

Se pot modifica specificațiile coloanelor utilizând comanda **ALTER TABLE**, cu clauza **MODIFY**. Modificările permise pot fi: schimbarea tipului de date, a mărimii și a valorii inițiale.

EXEMPLU



```
ALTER TABLE dept30
MODIFY (first_name varchar2(25));
Table altered.
```



Observații:

- Se poate mari lățimea sau precizia unei coloane numerice;
- Se poate micșora lățimea unei coloane dacă aceasta conține numai valori nule sau dacă tabela nu are rânduri;
- Se poate schimba tipul de date dacă coloana conține numai valori null;

- Se poate converti o coloană de tip CHAR la una de tip VARCHAR2 sau invers dacă aceasta conține valori null sau dacă nu se schimbă lățimea ;
- Schimbarea valorii predefinite pentru o coloană afectează numai inserările ulterioare în tabela.

Ștergerea unei coloane

EXEMPLU



```
ALTER TABLE dept30
DROP COLUMN job_id;
Table altered.
```

Observații:

- Coloana poate sau nu să conțină date ;
- Folosind aceasta comandă se poate șterge doar o singură coloană odată;
- Tabela trebuie să mai aibă cel puțin o coloană în urma ștergerii efectuate ;
- Odată ce coloana a fost ștersă, ea nu mai poate fi recuperată;

Opțiunea SET UNUSED

- Puteți folosi opțiunea **SET UNUSED** pentru a marca una sau mai multe coloane ca fiind nefolosite.
- Puteți folosi opțiunea **DROP UNUSED COLUMNS** pentru a șterge coloanele marcate ca fiind nefolosite.



```
ALTER TABLE table SET UNUSED (column);
sau
ALTER TABLE table SET UNUSED COLUMN (column);
```

```
ALTER TABLE table DROP UNUSED COLUMNS;
```

Opțiunea **SET UNUSED** marchează una sau mai multe coloane ca fiind nefolosite astfel încât ele să poată fi șterse atunci când resursele sistemului devin critice. Prin specificarea acestei clauze nu se șterg efectiv coloanele din fiecare rând ci ele sunt tratate ca și cum ar fi șterse. După ce coloana a fost marcată ca fiind nefolosită ea nu poate fi accesată, comanda **SELECT** nu va returna date din acest tip de coloană iar comanda **DESCRIBE** nu va afișa

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

numele și tipul de dată pentru coloanele marcate astfel. Puteți adăuga tabelei o nouă coloană cu același nume pe care îl are o coloană marcată ca fiind nefolositară.

Informația dată de SET UNUSED este stocată în dicționarul de date în view-ul **USER_UNUSED_COL_TABS**.

Opțiunea **DROP UNUSED** șterge din tabel coloanele marcate ca fiind nefolosite. Această opțiune se poate folosi atunci când se dorește eliberarea spațiului pe disc. Dacă tabela nu are coloane marcate, executarea instrucției nu întoarce eroare.

EXEMPLU



```
ALTER TABLE dept30  
SET UNUSED last_name;  
Table altered.
```

```
ALTER TABLE dept30  
DROP UNUSED COLUMNS;  
Table altered.
```

5.11 Stergerea unei tabele

Sintaxa:



```
DROP TABLE table;
```

EXEMPLU



```
SQL> DROP TABLE dept30;  
Table dropped.
```

Comanda **DROP TABLE** șterge definiția unei tabele din dicționarul de date Oracle. Când se aplică comanda **DROP** unei tabele, baza de date pierde toate înregistrările din tabelă, împreună cu indecșii asociați acesteia. **Comanda este ireversibilă.**



Observații:

- Toate datele sunt șterse;
- Orice view sau sinonim asociat va rămâne, dar va fi invalid;
- Orice tranzacție în curs va fi finalizată;
- Numai utilizatorul care a creat tabela sau cel care are privilegiul **DROP ANY TABLE** poate șterge o tabelă.

5.12 Modificarea numelui unui obiect

Pentru a modifica numele unei tabele, view, secvențe sau sinonim se folosește comanda **RENAME**.

Sintaxa:



```
RENAME old_name TO new_name;
```

- *old_name* numele vechi al obiectului;
- *new_name* numele nou al obiectului;

EXEMPLU



```
SQL> RENAME departments TO departments_new;  
Table renamed.
```



Observație: Numai proprietarul obiectului poate modifica numele obiectului.

5.13 Trunchierea unei tabele

Comanda șterge toate înregistrările din tabela specificată, eliberând spațiul folosit de tabelă. **Operațiunea este ireversibilă.**

Comanda **DELETE** șterge înregistrările din tabelă, dar nu eliberează spațiu de stocare (făcând astfel posibila readucerea înregistrărilor prin comanda **ROLLBACK**).



TRUNCATE TABLE table;

Sintaxa: - *table* este numele tabelei.

EXEMPLU



SQL>TRUNCATE TABLE department;
Table truncated.



Comanda **TRUNCATE** este mai rapidă decât **DELETE** din următoarele motive:

- **TRUNCATE** este o instrucțiune de tip DDL și nu generează informații de tip rollback.
- Trunchierea unei tabele nu declanșează triggerii tabelei. Dacă tabela este de tip părinte într-o relație de integritate referențială nu puteți aplica comanda **TRUNCATE**. Trebuie mai întâi să dezactivați constrângerea și apoi lansați comanda.

5.14 Includerea constrângerilor

Ce fac constrângerile ?

- Constrângerile forțează regulile la nivel de tabela.
- Constrângerile previn ștergerea unei tabele sau a datelor din tabela dacă există dependențe.

Oracle lucrează cu următoarele tipuri de constrângerii:

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

Constrângerile de integritate a datelor

Constrângere	Descriere
NOT NULL	Specifică faptul că această coloană nu poate conține o valoare nulă.
UNIQUE Key	Specifică o coloană sau o combinație de coloane a căror valoare trebuie să fie unică pentru toate înregistrările talei.
PRIMARY KEY	Identifică unic fiecare înregistrare
FOREIGN KEY	Stabilește și forțează o relație de tip cheie externă între coloană și o coloană dintr-o tabelă referită.
CHECK	Specifică o condiție care trebuie să fie adevărată.

Ghid pentru crearea constrângerilor

- Constrângerile trebuie să aibă un nume. Dacă utilizatorul nu dă acest nume, serverul Oracle va genera automat un nume utilizând formatul SYS_Cn, unde n este un număr întreg unic.
- Se poate crea o constrângere:
 - în timpul creării talelei sau
 - după ce tabela a fost creată
- Vizualizarea constrângerii se face doar în dicționarul de date.

Toate constrângerile sunt păstrate în dicționarul de date. Constrângerile sunt simplu de referit dacă li se dă un nume sugestiv. Constrângerile trebuie să urmeze regulile standard de denumire a obiectelor. Se pot vizualiza constrângerile create pentru o tabelă dacă accesăm din dicționarul view-ul **USER_CONSTRAINTS**.

Definirea constrângerilor



```
CREATE TABLE [schema.] table
  (column datatype [DEFAULT expr]
   (column_constraint),
   ...
   [table_constraint] [, ...]);
```

În sintaxa :

<i>schema</i>	același cu numele proprietarului
<i>table</i>	numele tablei
<i>DEFAULT expr</i>	specifică valoarea implicită
<i>column</i>	numele coloanei
<i>datatype</i>	tipul de dată și lungimea
<i>column_constraint</i>	constrângere de integritate ca parte a definiției coloanei
<i>table_constraint</i>	constrângere de integritate ca parte a definiției tablei

Exemplu:



```
CREATE TABLE employees(
    Employee_id number(6),
    First_name varchar2(20),
    ...
    Job_id varchar2(10) NOT NULL,
    CONSTRAINT emp_emp_id_pk
        PRIMARY KEY (EMPLOYEE_ID));
```

De obicei constrângerile sunt create în același timp cu tabela dar ele pot fi adăugate și după crearea tablei.

Constrângerile pot fi definite la unul din următoarele două nivele:

Constraint level	Descriere
Column	Referă o singură coloană și poate defini orice tip de constrângere
Table	Referă una sau mai multe coloane și este definită separat de definițiile coloanelor în tabela: poate defini orice tip de constrângere exceptând NOT NULL

```
column , ...
[CONSTRAINT constraint_name] constraint_type
(column, ...),
```

În sintaxă:

constraint_name este numele constrângerii
constraint_type este tipul constrângerii

5.15 Constrângerea NOT NULL

Constrângerea **NOT NULL** ne asigură că nu sunt permise în coloană valori **null**. Coloanele fără constrângerea **NOT NULL** pot conține implicit valori **null**.

EMPLOYEE_ID	FIRST_NAME	...	COMMIS SION_P CT	DEPARTMENT_ID
7839	KING			10
7698	BLAKE			30
7782	CLARK			10
7566	JONES			20
...				

Diagrama arată o tabelă cu cinci coloane: EMPLOYEE_ID, FIRST_NAME, ..., COMMIS_SION_PCT și DEPARTMENT_ID. Coloana ... este marcată cu un asterisk (...). Trei săgeți apointă spre această coloană. Săgeata din stânga susține textul "Constrângere NOT NULL (nici o înregistrare nu poate conține o valoare NULL pe aceasta coloană)". Săgeata din mijloc susține textul "Absența constrângerii NOT NULL (orice înregistrare poate conține null pentru aceasta coloană)". Săgeata din dreapta susține textul "Constrângere NOT NULL".

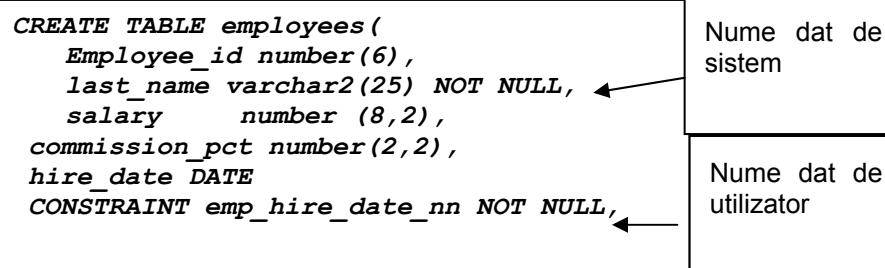
În exemplul de mai sus se aplică constrângerea **NOT NULL** coloanelor last_name și hire_date din tabela EMPLOYEES. Deoarece constrângerea pentru last_name nu are nume, Oracle Server va crea nume pentru ea.

OBSEVATIE:



Constrângerea **NOT NULL** poate fi specificată numai la nivel de coloană.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



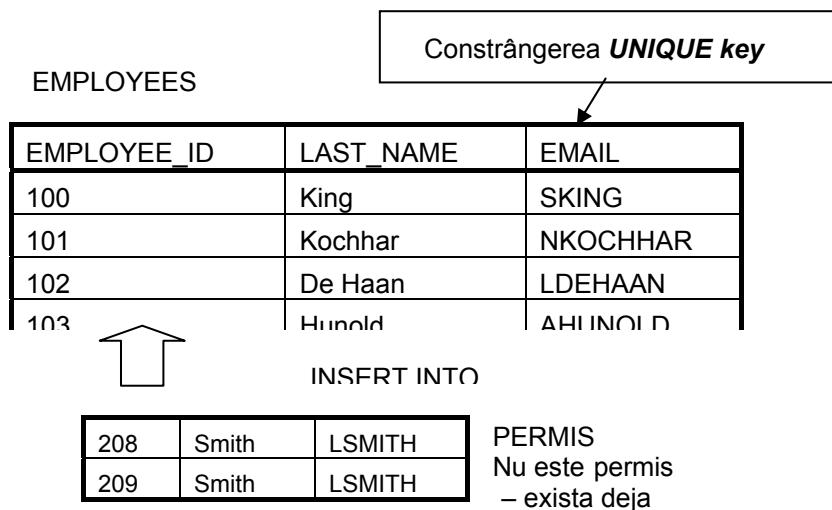
Numele constrângerii poate fi specificat în timpul specificării constrângerilor.



EXEMPLU:

*hire_date DATE
CONSTRAINT emp_hire_date_nn NOT NULL*

5.16 Constrângerea UNIQUE KEY



O constrângere de integritate de tip cheie unică cere ca fiecare valoare din coloana sau din setul de coloane specificat să fie unice – două înregistrări ale tablei nu pot avea valori dupicate în coloana sau setul de coloane care formează constrângerea. Coloana (setul de coloane) inclusă în definiția cheii unice se numește **cheie unică**. Dacă cheia unică conține mai multe coloane se

numește **cheie unică compusă**.

Constrângerea cheie unică permite introducerea de valori **null** dacă nu a fost definită o constrângere **NOT NULL** pentru acea coloană. De fapt, orice număr de înregistrări pot include valori null în coloane fără constrângerea **NOT NULL**. O valoare null într-o coloană (sau în toate coloanele unei chei unice compuse) satisfacă întotdeauna o constrângere de tip cheie unică.



```
CREATE TABLE employees (
    Employee_id number(6),
    last_name varchar2(25) NOT NULL,
    email varchar2(25),
    commission_pct number(2,2),
    hire_date DATE NOT NULL
    ...
CONSTRAINT emp_email_uk UNIQUE (email)
```

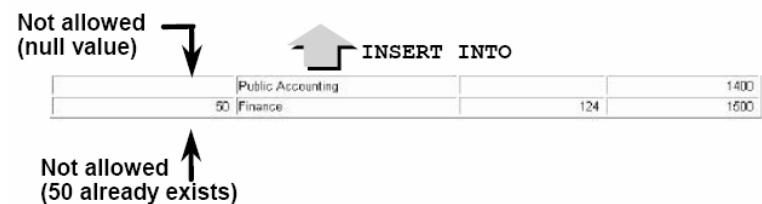


Constrângerea **cheie unică (unique key)** poate fi definită la nivel de tabelă sau coloană.

DEPARTMENTS

PRIMARY KEY

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1600
60	IT	103	1400
80	Sales	149	2500



În exemplu se aplică constrângerea de tip cheie unică coloanei EMAIL din tabela EMPLOYEES. Numele constrângerii este EMP_EMAIL_UK.



Notă:

O **cheie unică compusă este creată utilizând definiția la nivel de tabelă**. Serverul Oracle forțează implicit constrângerea de cheie unică creând un index unic după cheia unică.

5.17 Constrângerea PRIMARY KEY

Pentru fiecare tabelă poate fi creata doar o cheie primară. Constrângerea de tip cheie primară este formată dintr-o coloana sau set de coloane care identifică în mod unic fiecare înregistrare din tabelă. Aceasta constrângere forțează unicitatea coloanei sau a setului de coloane și asigură că nici o coloană care este parte a cheii primare nu poate conține o valoare null.

Un exemplu de constrângere în care cheia primara este formată din două coloane ar putea fi constrângerea pe coloanele serie și nr_buletin. Mai multe persoane pot avea aceeași serie de buletin dar nu pot avea același numărul de buletin. Același număr de buletin poate să apară la mai multe persoane, dar diferă seria și în același timp serie sau nr_buletin nu pot avea valori nule.

Notă:



- *Constrângerea de tip cheie primară poate fi definită la nivel de coloană sau tabelă.*
- *O cheie primară compusă este creată utilizând definiția la nivel de tabelă.*
- *Un index unic este automat creat pentru o coloană cheie primară.*

Exemplul definește o cheie primară după coloana DEPARTMENT_ID a tabelei DEPARTMENTS. Numele constrângerii este DEPT_ID_PK.

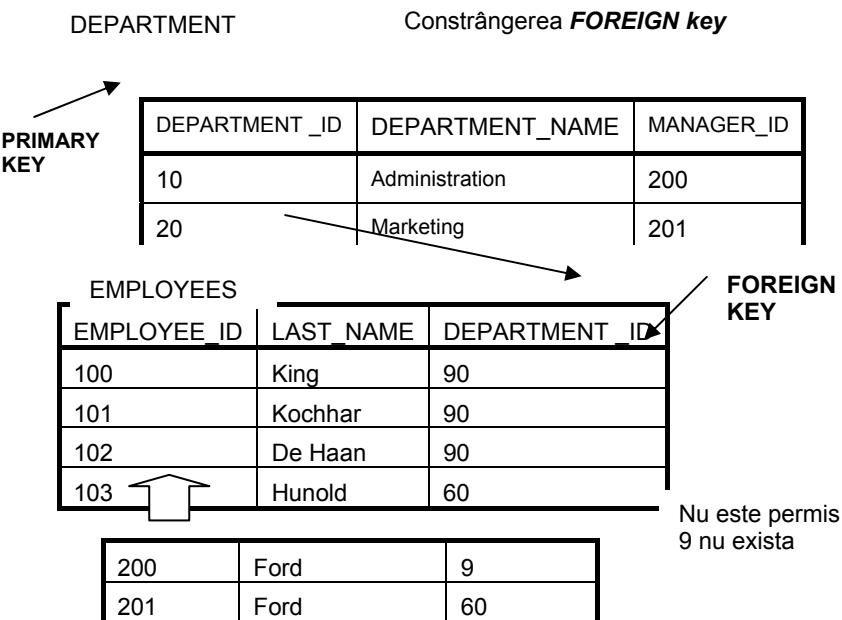
```
CREATE TABLE departments
  (department_id    NUMBER(4),
   department_name  VARCHAR2(30)
  CONSTRAINT dept_name_nn NOT NULL,
   manager_id       NUMBER(6),
   location_id      NUMBER(4),
  CONSTRAINT dept_id_pk PRIMARY KEY (department_id));
```

5.18 Constrângerea FOREIGN KEY

Cheia externă sau **constrângerea de integritate referențială**, desemnează o coloană sau o combinație de coloane în funcție de cheia externă și stabilește o relație cu o cheie primară sau o cheie unică în aceeași tabelă sau o tabela diferită. În exemplu, DEPARTMENT_ID a fost definit cheie externă în tabela EMPLOYEES (dependentă sau tabela copil); ea referă coloana DEPARTMENT_ID din tabela DEPARTMENTS (referită sau tabela părinte).

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Valoarea unei chei externe trebuie să se potrivească cu o valoare existentă în tabela părinte sau să fie **NULL**.



Observații:

- Cheile externe sunt bazate pe valorile datelor și sunt pointeri pur logici, nu fizici.
- Constrângerea de tip cheie externă poate fi definită la nivel de coloană sau tabelă.
- cheie externă compusă este creată folosind definiția la nivel de tabel.

Exemplul de mai sus definește o constrângere de tip cheie externă în coloana DEPARTMENT_ID din tabela EMPLOYEES. Numele constrângerii este EMP_DEPT_FK.



```
CREATE TABLE employees(
    Employee_id number(6),
    last_name varchar2(25) NOT NULL,
    email    varchar2(25),
    commission_pct number(2,2),
    hire_date DATE NOT NULL
    ...
    Department_id number (4),
    CONSTRAINT emp_dept_fk FOREIGN KEY
        (department_id) REFERENCES departments
        (department_id),CONSTRAINT emp_email_uk UNIQUE (email)
```

Cuvintele cheie ale constrângerii FOREIGN KEY

Constrângerile externe sunt definite în tabela copil și tabela care conține coloana referită este tabela părinte. Constrângerile externe sunt definite folosind combinații ale următoarelor cuvinte:

FOREIGN KEY- Definește coloana în tabela copil la nivelul constrângerii de tabel.

REFERENCES- Identifică tabela și coloana în tabela părinte.

ON DELETE CASCADE- Permite ștergeri în tabela părinte precum și ștergeri de linii independente în tabela copil. Când rândul din tabela părinte este șters, rândurile independente din tabela copil sunt deasemeni șterse. Fără opțiunea **ON DELETE CASCADE**, rândul din tabela părinte nu va putea fi șters dacă este referit în tabela copil.

ON DELETE SET NULL – convertește valoarea cheii străine pe **NULL** atunci când valoarea corespunzătoare din tabela părinte este ștearsa.

5.19 Constrângerea CHECK

Definește o condiție pe care fiecare rând trebuie să o îndeplinească.

Expresii care nu sunt permise:

- Referiri la pseudocoloanele **CURRVAL**, **NEXTVAL**, **LEVEL** și **ROWID**.
- Apeluri la funcțiile **SYSDATE**, **UID**, **USER** și **USERENV**.
- Interogări care fac referințe la alte valori din alte rânduri.

5.20 Adăugarea unei constrângeri



```
..., salary NUMBER (2)  
CONSTRAINT emp_salary_min CHECK (salary>0),...
```

Se poate adăuga o constrângere la tabelele existente folosind declarația **ALTER TABLE** împreună cu clauza **ADD**.

În sintaxă:

table este numele tablei



```
ALTER TABLE table  
ADD [CONSTRAINT constraint] type (column);
```

constraint este numele constrângerii

type este tipul constrângerii

column este numele coloanei afectate de constrângere

Denumirea constrângerii este optională și totuși ea este recomandată.

Dacă nu dați nume constrângerilor, sistemul va genera automat un nume.



- Se poate adăuga, șterge, activa, dezactiva o constrângere, dar nu se poate modifica structura acesteia.
- Se poate adăuga o constrângere de tip **NOT NULL** la o coloană existentă folosind clauza **MODIFY** din comanda **ALTER TABLE**.
- Se poate defini o coloană **NOT NULL** doar dacă tabela nu conține rânduri, deoarece nu se pot specifica date pentru rândurile existente în același timp în care adăugam noi coloane.

Exemplul de mai jos creează o constrângere de tip extern în tabela employees. Constrângerea ne asigură de existența manager-ului ca angajat activ în tabela employees.



```
ALTER TABLE employees  
ADD CONSTRAINT emp_manager_fk  
FOREIGN KEY (manager_id)  
REFERENCES employee (employee_id);
```

5.21 Ștergerea unei constrângerii

Exemplu : Ștergerea constrângerii manager din tabela EMPLOYEES.



```
ALTER TABLE employees
DROP CONSTRAINT emp_manager_fk;
Table altered.
```

Ștergerea constrângerii de tip cheie primara din tabela EMPLOYEES.

Pentru a șterge o constrângere trebuie identificat numele constrângerii, pe care îl puteți afla accesând **USER_CONSTRAINTS** și **USER_CONS_COLUMNS**. Apoi se folosește funcția **ALTER TABLE** împreună cu clauza **DROP**. Opțiunea **CASCADE** din clauza **DROP** are ca efect și eliminarea tuturor constrângerilor dependente.

```
ALTER TABLE employees
DROP PRIMARY KEY CASCADE;
Table altered.
```

Sintaxa:



```
ALTER TABLE table
DROP PRIMARY KEY | UNIQUE (column) |
CONSTRAINT constraint [CASCADE];
```

unde: table este numele tablei;
column este numele coloanei afectate de constrângere;
constraint este numele constrângerii.



Atenție:

Când se șterge o constrângere de integritate, aceasta nu mai este folosită de către Oracle Server și nu mai este disponibilă în dictionarul de date.

5.22 Dezactivarea constrângerilor

- Executarea clauzei **DISABLE** din funcția **ALTER TABLE** pentru a dezactiva o constrângere de integritate.
- Aplicarea opțiunii **CASCADE** pentru a dezactiva constrângerii de integritate dependente.



```
ALTER TABLE employees  
DISABLE CONSTRAINT emp_emp_id_pk CASCADE;  
Table altered.
```

Se poate dezactiva o constrângere fără a fi necesară ștergerea acesteia sau recrearea ei, folosind funcția **ALTER TABLE** împreună cu clauza **DISABLE**.



```
ALTER TABLE table  
DISABLE CONSTRAINT constraint [CASCADE];
```

Unde: table este numele tabelei.

constraint este numele constrângerii.

clauza **DISABLE** se poate folosi atât în funcția **CREATE TABLE** cât și în funcția **ALTER TABLE**.

clauza **CASCADE** dezactivează constrângerile de integritate dependente.

5.23 Activarea constrângerilor

- Activarea unei constrângerii de integritate care este dezactivată folosind clauza **ENABLE** în definirea tabelei.
- Un index de tip unic sau de tip cheie primară este automat creat dacă se activează constrângerile **UNIQUE** sau **PRIMARY KEY**.

Se poate activa o constrângere fără a o șterge sau a o recrea folosind funcția **ALTER TABLE** împreună cu clauza **ENABLE**.

Sintaxa:



```
ALTER TABLE table  
ENABLE CONSTRAINT constraint;
```

Unde: table este numele tabelei.

constraint este numele constrângerii.



```
ALTER TABLE employees  
ENABLE CONSTRAINT emp_emp_id_pk;  
Table altered.
```



Atenție:

- Daca se activează o constrângere, constrângerea este aplicată tuturor datelor din tabelă. Toate datele din tabelă trebuie să îndeplinească condițiile din constrângere.
- Daca se activează o constrângere de tip unic sau cheie primară, atunci este creat în mod automat un index de tip unic.
- Clauza **ENABLE** se poate folosi atât în funcția **CREATE TABLE** cât și în funcția **ALTER TABLE**.

Vizualizarea constrângerilor

Vizualizarea constrângerilor se poate face prin interogarea tabelei **USER_CONSTRAINTS** pentru a putea afla toate numele și definițiile constrângerilor.

CONSTRAINT_NAME	C	SEARCH_CONDITION
EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL
EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL
EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL
EMP_JOB_NN	C	"JOB_ID" IS NOT NULL
EMP_SALARY_MIN	C	salary > 0
EMP_EMAIL_UK	U	
EMP_EMP_ID_PK	P	
EMP_DEPT_FK	R	

După crearea unei tabele, se poate confirma existența sa prin folosirea comenzi **DESCRIBE**. Singura constrângere care poate fi verificată este constrângerea **NOT NULL**. Pentru a vedea toate constrângerile din tabelă, este necesară interogarea tabelei **USER_CONSTRAINTS**.



```
SELECT constraint_name, constraint_type,
       search_condition
  FROM user_cons_columns
 WHERE table_name='EMPLOYEES';
```

Exemplul afișează toate constrângerile tabelei EMPLOYEES.

NOTĂ:



Constrângerilor care nu primesc un nume de la posesorul tablei li se atribuie automat de către sistem un nume. La numirea tipului de constrângere, **C** provine de la **CHECK**, **P** de la **PRIMARY KEY**, **R** de la integritate referențială, și **U** de la **UNIQUE**. De observat faptul că constrângerea **NULL** este de fapt o constrângere de tip **CHECK**.

Vizualizarea coloanelor asociate constrângerilor

Vizualizarea coloanelor asociate cu numele constrângerilor se face folosind **USER_CONS_COLUMNS**. Aceasta vizualizare este utilă mai ales în cazul constrângerilor asociate de către sistem.



```
SELECT constraint_name, column_name  
FROM user_cons_columns  
WHERE table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	COLUMN_NAME
EMP_DEPT_FK	DEPARTMENT_ID
EMP_EMAIL_NN	EMAIL
EMP_EMAIL_UK	EMAIL
EMP_EMP_ID_PK	EMPLOYEE_ID
EMP_HIRE_DATE_NN	HIRE_DATE
EMP_JOB_FK	JOB_ID
EMP_JOB_NN	JOB_ID
EMP_LAST_NAME_NN	LAST_NAME
EMP_MANAGER_FK	MANAGER_ID
EMP_SALARY_MIN	SALARY



5.24 Exerciții

1. Creați tabelul DEPT bazat pe structura din tabelul următor. Salvați instrucțiunea în scriptul p1.sql , executați scriptul și verificați.

Column Name	Id	Name
Datatype	Number	Varchar2
Length	7	25

2. Introduceți înregistrări în tabela DEPT din tabela DEPARTMENT. Includeți doar coloanele de care aveți nevoie.
3. Creați tabela EMP bazată pe tabelul următor. Salvați instrucțiunea într-un script p3.sql și apoi executați scriptul. Verificați existența tăbelei.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Datatype	Number	Varchar2	Varchar2	Number
Length	7	25	25	7

4. Modificați tabela EMP pentru a permite nume mai lungi în coloana Last_name. Verificați efectuarea modificării.
5. Asigurați-vă că cele două tabele create sunt stocate de dicționarul de date.
6. Creați tabela EMPLOYEE2 bazată pe structura tăbelei EMP, inclusiv doar coloanele EMPNO, ENAME și DEPTNO. Redenumiți coloanele în noua tăbelă astfel: ID, LAST_NAME, DEPT_ID.
7. Redenumiți tăbelă EMPLOYEE2 în EMPLOYEE3.
8. Stergeți tăbelă EMPLOYEE3.
9. Adăugați o constrângere de tip PRIMARY KEY la tăbelă EMP folosind coloana ID. Constrângerea trebuie să fie activată la creare.
10. Creați o constrângere PRIMARY KEY în tăbelă DEPT folosind coloana ID. Constrângerea trebuie să fie activată la creare.
11. Adăugați o referință de tip cheie externă la tăbelă EMP care va asigura ca angajații nu sunt asignați unui departament inexistent.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

12. Conformați constrângerile adăugate interogând tabela USER_CONSTRAINTS. Observați numele și tipul constrângerilor. Salvați textul într-un fișier.

CONSTRAINT_NAME	C
DEPT_ID_PK	P
EMP_ID_PK	P
EMP_DEPT_ID_FK	R

13. Afisați numele și tipul obiectelor din dicționarul USER_OBJECTS al tabelelor EMP și DEPT. Formatați coloanele pentru reutilizarea lor. Observați că a fost creat un nou tabel precum și un nou index.

OBJECT_NAME	OBJECT_TYPE
DEPT	TABLE
DEPT_ID_PK	INDEX
EMP	TABLE
EMP_ID_PK	INDEX

14. Modificați tabela EMP. Adăugați o coloană SALARY de tip NUMBER, precizie 7.
15. Stergeți coloana FIRST_NAME din tabela EMP. Conformați modificările efectuate prin verificarea descrierii tăbelei.
16. Marcați ca fiind nefolosită coloana DEPT_ID din tabela EMP. Conformați modificările efectuate prin verificarea descrierii tăbelei.
17. Stergeți toate coloanele nefolosite din tabela EMP. Conformați modificările efectuate prin verificarea descrierii tăbelei.

Capitolul 6 Expresii aritmetice. Operatori. Restrictionarea și sortarea datelor

Instrucțiunea **SELECT** poate fi folosită pentru a afișa anumite coloane din tabela specificând numele coloanelor, separate prin virgulă. În clauzele **SELECT** se specifică coloanele dorite, în ordinea de afișare dorită.

Modul implicit de afișare a informațiilor este :

- Stânga – date calendaristice și caractere (exemplu: coloanele *ename* și *hiredate*)
- Dreapta – date numerice (exemplu: coloana *sal*)
- Textul este scris cu litere mari (uppercase)

Numele de coloană pentru date calendaristice sau caractere poate fi trunchiat, dar numele de coloană pentru datele de tip numeric nu poate fi trunchiat. Numele coloanelor sunt afișate implicit cu litere mari. Titlul coloanelor poate fi modificat folosind un alias. Folosirea alias-urilor va fi prezentată într-un capitol ulterior.

6.1 Expresii aritmetice

Crearea expresiilor numerice și de date folosind operatori aritmetici modifică felul în care se afișează datele prin executarea de calcule. Expresiile aritmetice pot conține nume de coloane, constante numerice și operatori aritmetici.

Operatorii aritmetici disponibili în SQL se pot folosi în orice clauza SQL exceptând clauza FROM.

Operator	Descriere
+	Adunare
-	Scădere
*	Înmulțire
/	Împărțire

 **Exemplu:**

SELECT last_name, salary, salary+300 FROM employees;

În exemplul dat s-a folosit operatorul *adunare* pentru a mări salariile cu 300\$ pentru toți angajații. Se afișează noua coloană SALARY+300 . În urma adunării coloana rezultat (SALARY+300) nu este o coloană nouă în tabela EMPLOYEES; aceasta este vizibilă doar la afișarea rezultatelor. Implicit,

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

denumirea noii coloane provine de la operația care a generat-o, în acest caz SALARY+300.



NOTĂ: SQL*Plus ignoră spațiile din față și din spatele operatorilor aritmetici.

Prioritatea operatorilor este * / + -

- Înmulțirea și împărțirea au prioritate față de adunare și scădere.
- Operatorii de aceeași prioritate sunt evaluati de la stânga la dreapta.
- Parantezele sunt folosite pentru a forța evaluările prioritare și a clarifica regulile.
- Dacă o expresie aritmetică conține mai mult de un operator, înmulțirea și împărțirea sunt evaluate primele. Dacă operatorii folosiți într-o expresie sunt de aceeași prioritate, evaluarea se va face de la stânga la dreapta. Puteți folosi parantezele pentru a forța expresia din paranteze să fie evaluată prima.



Exemplu:

```
SELECT last_name, salary, 12*salary+100 FROM employees;
```

LAST_NAME	SALARY	12*SALARY+100
King	24000	288100
Kochhar	17000	204100
De Haan	17000	204100
Hunold	9000	108100
Ernest	6000	72100
Lorentz	4200	50500

```
Gietz _____ | 8300 | _____ | 99700 |
```

20 rows selected.

În exemplul dat sunt afișate câmpurile nume, salariu și compensația anuală a angajaților. Aceasta este calculată înmulțind salariul lunar cu 12 plus o primă de 100\$. Deci, înmulțirea se efectuează înaintea adunării.

Folosirea parantezelor pentru a întări ordinea firească a operațiilor va arata astfel : $(12*\text{SALARY})+100$, operație care nu va schimba rezultatul.

Folosirea parantezelor



Exemplu:

SELECT last_name, salary, 12*(salary+100) FROM employees;

Se pot modifica regulile precedentei operatorilor folosind parantezele

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200
Lorentz	4200	51600

Gietz | 8300 | 100800

20 rows selected.

pentru a specifica ordinea în care operatorii să fie folosiți. În exemplul dat sunt afișate numele, salariul și compensația anuală a salariaților. Aceasta este formată din salariul lunar plus o prima, totul înmulțit cu 12. Datorită parantezelor, adunarea are prioritate față de înmulțire.

Definirea valorii nule (null value)



Valoarea null este nedisponibilă, neatribuită, necunoscută sau inaplicabilă.

Valoarea nul nu este aceeași cu zero sau spațiu.

Exemplu:



***SELECT last_name, job_id, salary, commission_pct
FROM employees;***

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
King	AD_PRES	24000	
Kochhar	AD_VP	17000	
Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8800	.2
Montgomery	SA_MAN	12000	
Gietz	AC_ACCOUNT	8000	

20 rows selected.

Dacă o linie nu are date pentru o coloană anume, aceasta valoare se numește nulă.

Valoarea **null** este nedisponibilă, neatribuită, necunoscută sau inaplicabilă. Valoarea **null** nu este aceeași cu zero sau spațiu. Zero este număr iar spațiul este un caracter.

Coloanele de orice tip de dată pot conține valoarea vidă, cu excepția celor care au fost definite nenule sau chei primare.

Se observă că unii angajați pot câștiga comision iar alții nu - coloana COMMISSION_PCT din tabela EMPLOYEES. Valoarea vidă reprezintă un fapt în sine. Trebuie subliniat faptul ca vânzătorul are comisionul 0 nu null.

Valoarea nulă în expresii aritmetice



Expresiile aritmetice care conțin valoarea null sunt evaluate ca nule.



Exemplu:

```
SELECT last_name, 12*salary*commission_pct  
FROM employees;
```

Dacă o coloană dintr-o expresie aritmetică conține valoarea null, rezultatul este **null**. De exemplu, dacă încercați să executați o împărțire la zero, obțineți o eroare. Oricum, dacă împărțiți un număr la valoarea null, rezultatul este null sau necunoscut.

Facultatea de Automatică și Calculatoare Iași **Baze de date – lucrări practice**

LAST_NAME	12*SALARY*COMMISSION_PCT
King	
Kochhar	
Zlotkey	25200
Abel	30600
Taylor	20640
Higgins	
Gietz	

20 rows selected.

În exemplul de mai sus, angajatul KING nu primește comision. Deoarece coloana COMMISSION_PCT în expresia aritmetică este nulă, rezultatul este null.

6.2 Definirea alias-urilor pentru coloane

Aliasurile redenumesc numele coloanei.

Aliasurile sunt folositoare acolo unde se folosesc calcule aritmetice.

- Aliasurile urmează imediat după numele coloanei; optional se poate folosi cuvântul cheie **AS** între numele coloanei și alias;
- Aliasurile necesită două ghilimele (“ ” / dacă conțin spații, caractere speciale sau au importanță literale mari (mici).

La afișarea rezultatului unei interogări SQL* Plus folosește în mod normal numele coloanei drept cap de tabel. În multe cazuri acest cap de tabel nu este sugestiv și de aici dificultatea de a înțelege conținutul coloanei. Numele coloanei se poate schimba folosind aliasul.

Aliasul se specifică după numele coloanei în lista SELECT folosind spațiul ca separator. Implicit capul de tabel obținut prin alias este scris cu litere mari. Dacă aliasul conține spații, caractere speciale (ca \$ sau #), sau au importanță literale mari (mici), aliasul trebuie scris între ghilimele (“”).



Exemplu:

```
SELECT last_name as name, commission_pct comm  
FROM employees;
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

NAME	COMM
King	
Kochhar	
Higgins	
Gietz	

20 rows selected.

**SELECT last_name as „Name”, salary*12 „Annual Salary”
FROM employees;**

Name	Annual Salary
King	288000
Kochhar	204000
Higgins	144000
Gietz	99600

20 rows selected.

Primul exemplu afișează numele și comisionul lunar al tuturor angajaților. Cuvântul cheie AS a fost folosit înainte de alias. Rezultatul interogării trebuie să fie același indiferent dacă cuvântul cheie AS este folosit sau nu. Un alt aspect ce trebuie remarcat este faptul că name și comm au fost scrise cu litere mici iar afișarea s-a făcut cu litere mari. Deci, implicit capul de coloana apare cu litere mari.

În cel de-al doilea exemplu afișăm numele și salariul anual al angajaților. Deoarece Annual Salary implică folosirea spațiului, aliasul trebuie scris între ghilimele. Astfel va fi afișat exact ceea ce este scris în expresia SELECT.

6.3 Operatorul de concatenare

- Concatenează coloanele sau sirurile de caractere cu alte coloane;
- Este reprezentat de două bare verticale (||);
- Rezultă o coloană care este o expresie caracter.

Folosind operatorul de concatenare (||) se pot concatena coloane, expresii aritmetice sau valori constante pentru a crea expresii de tip caracter. Coloanele situate de o parte și de alta a operatorului sunt combinate pentru a face o singură coloană de ieșire.



Exemplu:

SELECT last_name || job_id as 'Employees' FROM employees;

Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP
HunoldIT_PROG

GietzAC_ACCOUNT

20 rows selected

În exemplul de mai sus **last_name și job_id** sunt concatenate având aliasul **Employees**. Observați că funcția angajatului și numele acestuia sunt combinate pentru a rezulta o singură coloana de ieșire. Cuvântul cheie AS, folosit înaintea numelui aliasului, face mai ușor de citit instrucțiunea SELECT.



Şiruri de caractere

- Un “literal” este un caracter, o expresie sau un număr inclus în lista SELECT;
- Valorile literale pentru datele calendaristice și caractere trebuie incluse între ghilimele simple;
- Fiecare sir de caractere este afișat odată pentru fiecare rând întors.



Exemplu:

**SELECT last_name || ' is a ' || job_id as "Employee Details"
FROM employees;**

Exemplul afișează numele și meseriile tuturor angajaților. Coloana poartă titlul Detaliile angajatului. Observați spațiile dintre ghilimele simple din instrucțiunea SELECT care îmbunătățesc lizibilitatea ieșirii.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG

Gietz is a AC_ACCOUNT

20 rows selected.

În exemplul următor, numele și salariul fiecărui angajat este concatenat cu un literal pentru a da rândurilor mai mult înțeles.



***SELECT last_name ||':1 Month salary = '||salary Monthly
FROM employees;***

MONTHLY
King: 1 Month salary = 24000
Kochhar: 1 Month salary = 17000
De Haan: 1 Month salary = 17000
Hunold: 1 Month salary = 9000
Ernst: 1 Month salary = 6000
Lorentz: 1 Month salary = 4200
Mourgos: 1 Month salary = 5800
Rajs: 1 Month salary = 3500

20 rows selected.

Rânduri duplicate

Implicit, interogările afișează toate rândurile, inclusiv rândurile duplicate.



Exemplu:
SELECT department_id FROM employees;

DEPARTMENT_ID
90
90
90
60
60
60
60
60
60

20 rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

DEPARTMENT_ID	JOB_ID
10	AD_ASST
20	MK_MAN
20	MK_REP
50	ST_CLERK
50	ST_MAN
60	IT_PROG
80	SA_MAN
80	SA_REP

13 rows selected.

SQL*Plus va afișa rezultatul interogării fără a elimina rândurile duplicate dacă nu se indică altfel. Exemplul de mai sus afișează toate numerele de departamente din tabelul EMP.

Eliminarea rândurilor duplicate se face folosind cuvântul cheie *DISTINCT* în clauza *SELECT*.



Exemplu:

DEPARTMENT_ID
10
20
50
60
80
90
110

8 rows selected.

SELECT DISTINCT department_id FROM employees;

Se pot specifica mai multe coloane după clauza DISTINCT. Această cluză afectează toate coloanele selectate și rezultatul reprezintă o combinație de coloane distincte.

SELECT DISTINCT department_id , job_id FROM employees;

6.4 Afișarea structurii unei tabele

Se folosește comanda SQL*Plus:

DESC [RIBE] tablename

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

unde tablename este numele unei tabele existente, a unei vizualizări sau a unui sinonim accesibil utilizatorului.



Exemplu:
DESCRIBE EMPLOYEES

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Se afișează informații despre structura tabelei **EMPLOYEES**. Null? indică dacă o coloană trebuie să conțină date; NOT NULL indică faptul că acea coloană trebuie să conțină date. Type afișează tipul de dată al coloanei.

Tip de date	Descriere
NUMBER(p,s)	Valori numerice având un număr maxim de p cifre, unde s este numărul de cifre din dreapta virgulei
VARCHAR2(s)	Șir de caractere de lungime variabilă cu lungime maxima s
DATE	Dată calendaristică între 1 ianuarie 4712 i.c. și 31 decembrie 9999 d.c.
CHAR(s)	Șir de caractere de lungime fixă (s)

6.5 Restricționarea și sortarea datelor

La citirea datelor dintr-o baza de date s-ar putea să fie necesară reducerea numărului de linii afișate sau specificarea ordinii în care acestea să fie afișate.

În exemplul de mai jos se dorește afișarea tuturor angajaților din departamentul 90. Setul de linii care au valoarea 90 în coloana DEPARTMENT_ID sunt singurele returnate. Această metodă de restricționare reprezintă baza clauzei **WHERE** în SQL.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Limitarea liniilor folosind o selecție

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	...	DEPARTMENT_ID
100	King	AD_PRES		90
101	Kochhar	AD_VP		90
102	De Haan	AD_VP		90
103	Hunold	IT_PROG		90
...				60

“... returnează
toți angajații
din
departamentul
90”

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	...	DEPARTMENT_ID
100	King	AD_PRES		90
101	Kochhar	AD_VP		90
102	De Haan	AD_VP		90

6.6 Clauza WHERE



```
SELECT [DISTINCT] {*, column [alias], ...}
FROM table
[WHERE condition(s)];
```

În sintaxă:

WHERE *condition* restricționează interogarea la liniile ce îndeplinesc condiția.
e compusă din nume de coloane, expresii, constante și operatori de comparație.

Clauza WHERE urmează după clauza FROM

Puteți reduce numărul de linii returnate de o interogare folosind clauza **WHERE**.

O clauza **WHERE** conține o condiție ce trebuie îndeplinită și urmează imediat după o clauza **FROM**.

Clauza **WHERE** poate compara valorile din coloane, valori literale, expresii aritmetice sau funcții. Clauza **WHERE** e compusă din trei elemente ;

- Numele coloanei;
- Operatorul de comparație;
- Nume de coloana, constantă sau listă de valori.



Exemplu:

```
SELECT employee_id, last_name, first_name, department_id  
FROM employees WHERE department_id=90;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

În exemplul de mai sus SELECT-ul returnează numele și funcția tuturor angajaților din departamentul 90.

Şiruri de caractere și date



- Şirurile de caractere și datele sunt incluse între apostrof;
- Valorile de tip caracter sunt **case-sensitiv** iar valorile de tip dată calendaristică sunt **format-sensitive**;
- Formatul implicit pentru dată calendaristică este 'DD-MON-YY'



Exemplu:

```
SELECT last_name, job_id, department_id FROM employees  
WHERE last_name='Whalen';
```

LAST_NAME	JOB_ID	DEPARTMENT_ID
Whalen	AD_ASST	10

Şirurile de caractere și datele din clauza **WHERE** trebuie incluse între apostrofuri (' '). Regula nu se aplică constantelor numerice. Toate căutările de tip caracter sunt **case-sensitiv**. În exemplul următor nu este returnată nici o linie deoarece tabela EMPLOYEES conține toate datele scrise cu majuscule.



```
SELECT last_name, job_id, department_id  
FROM employees  
WHERE last_name='WHALEN';
```

Oracle retine datele într-un format numeric intern, reprezentând secol, an, luna, zi, ore, minute și secunde. Afisarea implicită a datei este DD-MON-YY.



NOTĂ: schimbarea formatului implicit va fi explicat în alt capitolul.

6.7 Operatori de comparație

Operator	Semnificație
=	Egal cu
>	Mai mare decât
>=	Mai mare sau egal
<	Mai mic decât
<=	Mai mic sau egal
<>	Diferit de
!=	Diferit de

Operatorii de comparație sunt folosiți în condițiile care compară două expresii. Aceștia sunt folosiți în clauza **WHERE** în următorul format:



Sintaxa:

WHERE expresie operator valoare



Exemplu:

**WHERE hiredate='01-JAN-95'
WHERE sal>=1500
WHERE ename='SMITH'**

Folosirea operatorilor de comparație

```
SELECT last_name, salary  
FROM employees  
WHERE salary<=3000;
```

În exemplul de mai sus SELECT-ul returnează numele și salariul din tabela EMPLOYEES acolo unde salariul angajatului este mai mic sau egal cu 3000. Cele două valori sunt luate din coloanele SALARY respectiv LAST_NAME ale tabelei EMPLOYEES.

Alți operatori de comparație

Operator	Semnificație
BETWEEN ...AND...	Între două valori (inclusiv)
IN(lista)	Potrivește orice valoare din listă
LIKE	Potrivește un tip de caracter
IS NULL	Este o valoare nulă

Operatorul **BETWEEN**

Operatorul **BETWEEN** se folosește pentru afișarea valorilor cuprinse într-un interval.



```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```

Limita Inferioară Limita Superioară



Folosind operatorul **BETWEEN** liniile afișate vor avea valori cuprinse în intervalul cuprins între limita inferioară și limita superioară.

Valorile specificate cu operatorul **BETWEEN sunt inclusive.** Prima data trebuie specificată limita inferioară.

SELECT-ul de mai sus returnează liniile din tabela EMPLOYEES pentru toți angajații căror salar este cuprins între 2500 și 3500.

Folosirea operatorului **IN la testarea valorilor dintr-o listă.**



```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100,101,201);
```

Exemplul de mai sus afișează numărul, numele, salariul și numărul managerului pentru toți angajații ai căror manageri au numărul 100,101 sau 201.

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Muurgos	5800	100
149	Zlotkey	10500	100
201	Harstein	13000	100

8 rows selected.

Operatorul **IN** poate fi folosit cu orice tip de dată.

Următorul exemplu returnează o linie din tabela EMPLOYEES pentru fiecare angajat al căruia nume este inclus în lista de nume din clauza WHERE.



```
SELECT employee_id,manager_id,department_id  
FROM employees  
WHERE last_name IN ('Harstein','Vargas');
```

Dacă în listă sunt folosite caractere sau date acestea trebuie incluse între apostrof.

Folosirea operatorului **LIKE**

Operatorul **LIKE** se folosește pentru a efectua căutări folosind caractere *wildcard* în sirurile valide de căutare.

Condițiile căutării pot conține fie caractere literale, fie numere.

- % înseamnă zero sau mai multe caractere
- _ înseamnă un singur caracter



```
SELECT first_name  
FROM EMPLOYEES  
WHERE first_name LIKE 'S%';
```

Nu cunoaștem întotdeauna exact valoarea pe care o căutăm. Folosind operatorul **LIKE** se pot selecta liniile care se potrivesc cu un anumit şablon de cătare format dintr-un sir de caractere numit *wildcard*. Pentru construirea sirurilor de căutare pot fi folosite două simboluri.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

EMPLOYEE_ID	LAST_NAME	JOB_ID
149	Zlotkey	SA_MAN
174	Abel	SA_REP
176	Taylor	SA_REP
178	Grant	SA_REP

Simbol	Semnificație
%	Reprezintă orice secvență de două sau mai multe caractere
_	Reprezintă un singur caracter

SELECT-ul de mai sus afișează numele angajaților din tabela EMPLOYEES pentru toți angajații ai căror nume începe cu "S". Numele care încep cu "s" nu vor fi returnate.

Operatorul **LIKE** poate fi folosit și în comparațiile ce folosesc operatorul **BETWEEN**. Următorul exemplu afișează numele și data angajării pentru salariații ce au fost angajați între ianuarie 1981 și decembrie 1981.

***SELECT last_name, hire_date FROM EMPLOYEES
WHERE hire_date LIKE '%95';***

Se pot face diverse combinații de caractere cum ar fi :



***SELECT last_name FROM EMPLOYEES
WHERE last_name LIKE '_o%';***

Se poate folosi identificatorul **ESCAPE** pentru a căuta caracterele "%" și "_".

Combinări de caractere wildcard:

Simbolurile % și _ pot fi folosite în orice combinație cu caracterele literale. Exemplul afișează numele tuturor angajaților care au al doilea caracter "A".

Opțiunea ESCAPE:

Când este nevoie de o potrivire exactă a caracterelor "%" și "_" trebuie folosita opțiunea ESCAPE. Aceasta opțiune specifică care este caracterul ESCAPE. Pentru a afișa numele tuturor angajaților care conține secvența "A_B" se va folosi următorul SELECT:



***SELECT employee_id, last_name, job_id
FROM EMPLOYEES
WHERE job_id LIKE '%SA_%' ESCAPE '\';***

Facultatea de Automatică și Calculatoare Iași **Baze de date – lucrări practice**

Opțiunea ESCAPE identifică caracterul backslash (\) ca fiind caracter ESCAPE. În tipar caracterul ESCAPE precedă caracterul underscore (_). Acest lucru face ca Oracle să interpreteze caracterul underscore drept literal.

Folosirea operatorului *IS NULL* la testarea valorilor nule



```
SELECT last_name,manager_id  
FROM EMPLOYEES  
WHERE manager_id IS NULL;
```

Operatorul **IS NULL** căuta valorile nule. O valoare nulă este o valoare nedisponibilă, neatribuită, necunoscută sau neaplicabilă. Din această cauză nu poate fi testată folosind operatorul (=). O valoare nulă nu poate fi egală sau inegală cu orice valoare. Exemplul de mai sus returnează numele și managerul tuturor angajaților care nu au manager.

De exemplu pentru a afișa numele, funcția și comisionul tuturor angajaților care nu au dreptul la comision se va folosi următorul SELECT:



```
SELECT last_name,job_id,commission_pct  
FROM EMPLOYEES  
WHERE commission_pct IS NULL;
```

LAST_NAME	JOB_ID	COMMISSION_PCT
King	AD_PRES	
Kochhar	AD_VP	
De Haan	AD_VP	
Gietz	AC_ACCOUNT	

16 rows selected.

6.8 Operatori logici

Operator	Semnificație
AND	Returnează TRUE dacă ambele componente ale condiției sunt adevărate
OR	Returnează TRUE dacă una din componentele condiției este adevărată
NOT	Returnează TRUE dacă condiția este falsă

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Un operator logic combină rezultatul a două componente de tip condiție pentru a produce un singur rezultat bazat pe acestea sau pentru a inversa rezultatul unei singure condiții.

Folosirea operatorului AND

AND cere ca ambele condiții sa fie adevărate.



```
SELECT employee_id, last_name, job_id, salary  
FROM EMPLOYEES  
WHERE salary >= 10000 AND job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

În exemplul de mai sus ambele condiții trebuie să fie adevărate pentru a fi selectată vreo înregistrare. De acea va fi selectat doar angajatul pentru care numele codului funcției începe cu MAN și care câștigă mai mult de 10000. Toate căutările de tip caracter sunt *case-sensitive*. Nu va fi returnată nici o linie dacă MAN nu este scris cu majuscule. Sirurile tip caracter trebuie incluse între apostrof.

Tabela de adevăr a operatorului AND



AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

Folosirea operatorului OR

Operatorul OR cere ca doar una din condiții sa fie adevărată.



```
SELECT employee_id, last_name, job_id, salary  
FROM EMPLOYEES  
WHERE salary >= 10000 OR job_id LIKE '%MAN%';
```

În exemplul de mai sus oricare din condiții poate fi adevărată pentru a fi selectată vreo înregistrare. De acea un angajat pentru care numele codului funcției începe cu MAN sau câștiga mai mult de 10000 va fi selectat.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

8 rows selected.

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

10 rows selected.

Tabela de adevăr a operatorului OR



OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Folosirea operatorului NOT



```
SELECT last_name,job_id,salary
FROM EMPLOYEES
WHERE job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

În exemplul de mai sus sunt afișate numele și funcția tuturor angajaților a căror funcție **nu este IT_PROG, ST_CLERK, SA_REP**.

Tabela de adevăr a operatorului NOT



NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN



Notă: operatorul **NOT** poate fi folosit împreună cu alți operatori SQL cum ar fi : **BETWEEN**, **LIKE** și **NULL**.



... *WHERE job_id NOT in ('AC_ACCOUNT','AD_VP')*
 ... *WHERE salary NOT BETWEEN 10000 AND 15000*
 ... *WHERE last_name LIKE '%A%'*
 ... *WHERE commission_pct IS NOT NULL*

Reguli de precedență



Ordinea evaluării	Operator
1	Toți operatorii de comparație
2	Operatori de concatenare
3	Condiții de comparare
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT
7	AND
8	OR



Precedența regulilor se poate schimba folosind paranteze.



**SELECT last_name,job_id,salary FROM EMPLOYEES
 WHERE job_id ='SA_REP'
 OR JOB_ID='AD_PRES' AND SALARY>15000;**

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	5600
Grant	SA_REP	7000

În exemplul de mai sus sunt două condiții ce trebuie înndeplinite și anume:

- Funcția trebuie să fie AD_PRES și salariul sa fie mai mare de 15000.
- Funcția trebuie să fie SA_REP.

Facultatea de Automatică și Calculatoare Iași **Baze de date – lucrări practice**

De aceea SELECT-ul se citește după cum urmează: "Selectează linia dacă un angajat este AD_PRES și câștiga mai mult de 15000 **sau** dacă angajatul este SA_REP."



Folosirea parantezelor pentru a forța prioritatea:

```
SELECT last_name,job_id,salary FROM EMPLOYEES
WHERE(job_id='SA_REP' OR JOB_ID='AD_PRES')AND
SALARY>15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

În exemplul de mai sus sunt două condiții:

- Funcția trebuie să fie SA_REP **sau** AD_PRES ;
- Salariul trebuie să fie mai mare de 15000.

De aceea SELECT-ul citește următoarele: selectează linia dacă angajatul este AD_PRES sau SA_REP și dacă angajatul câștiga mai mult de 15000."

6.9 Clauza ORDER BY

Ordinea liniilor returnate de o interogare nu este predefinită. Clauza **ORDER BY** poate fi folosită pentru a sorta liniile. Dacă este folosită clauza **ORDER BY**, ea trebuie scrisă ultima în fraza **SELECT**. Se poate specifica sortarea după o expresie sau după un alias.

Sintaxa:



```
SELECT expresie
FROM table
[WHERE condition(s)]
[ORDER BY {coloana, expresie} ASC|DESC];
```

unde: ORDER BY specifică ordinea în care sunt afișate liniile.
 ASC ordonează liniile ascendent – implicit.
 DESC ordonează liniile descendent.

Dacă nu este folosită clauza **ORDER BY** ordinea sortării este nedefinită și este posibil ca Serverul Oracle să nu afișeze liniile în aceeași ordine, pentru aceeași interogare, de două ori. Folosiți clauza **ORDER BY** pentru a afișa liniile într-o ordine specifică.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Sortarea în ordine descendenta



```
SELECT last_name,job_id,department_id, hire_date  
FROM EMPLOYEES  
ORDER BY hire_date DESC;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Zlotkey	SA_MAN	80	29-JAN-00
Mourgos	ST_MAN	50	16-NOV-99
Grant	SA_REP		24-MAY-99
Lorentz	IT_PROG	60	07-FEB-99
Vargas	ST_CLERK	50	09-JUL-98
Taylor	SA_REP	80	24-MAR-98
Matos	ST_CLERK	50	16-MAR-98
Fay	MK_REP	20	17-AUG-97
Davies	ST_CLERK	50	29-JAN-97
Abel	SA_REP	80	11-MAY-96
King	AD_PRES	90	17-JUN-87

20 rows selected.



La folosirea clauzei ORDER BY valoarea implicită este cea ascendentă.

- Valorile numerice sunt afişate începând cu cea mai mică valoare – de exemplu 1- 999.
- Datele sunt afişate începând cu cea mai timpurie – de exemplu 01-JAN-92 înaintea lui 01-JAN-95.
- Valorile tip caracter sunt afişate în ordine alfabetică – de exemplu A înaintea lui Z.
- Valorile nule sunt afişate ultimele pentru secvențe ascendente și primele pentru secvențe descendente.

Inversarea ordinii implicate în care sunt afişate liniile se face prin specificarea cuvântului cheie **DESC** după numele coloanei în clauza **ORDER BY**. În exemplu rezultatele sunt sortate după cea mai recentă dată de angajare a salariaților.

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

Sortarea după aliasul coloanei



**SELECT employee_id, last_name,salary*12 annsal
FROM EMPLOYEES ORDER BY annsal;**

EMPLOYEE_ID	LAST_NAME	ANNSAL
144	Vergas	30000
143	Mates	31200
142	Dawes	37200
141	Rajs	42000
107	Lorentz	50400
200	Whalen	52800
124	Moungos	69600
104	Ernst	72000
202	Fay	72000
178	Grant	84000
206	Gietz	96000
100	King	288000

20 rows selected.

Se poate folosi aliasul unei coloane în clauza **ORDER BY**. În exemplul de mai sus datele sunt sortate după salariul anual.

Sortarea după mai multe coloane

- Ordinea listei scrise în clauza **ORDER BY** este ordinea sortării.
- Se poate face sortare și după o coloană care nu este în lista SELECT.
- Rezultatele interogării pot fi sortate după mai multe coloane. Limita sortării este dată de numărul de coloane din tabela respectivă.
- În clauza **ORDER BY** trebuie specificate coloanele separate prin virgule. Dacă se dorește schimbarea ordinii afișării unei coloane se specifică **DESC** după numele coloanei respective.



Exemplu:

Să se afișeze numele și salariul tuturor angajaților. Rezultatele să se afișeze după numărul departamentului și apoi în ordine descrescătoare după salariu.

**SELECT last_name,department_id,salary
FROM EMPLOYEES ORDER BY department_id,salary DESC ;**

LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Moungos	50	5900
Rajs	50	3500
Higgins	110	12000
Gietz	110	8300
Grant		7000

20 rows selected.



6.10 Exerciții

1. Inițiați o sesiune SQL*Plus.
2. Comenzile SQL*Plus accesează o baza de date: adevărat / fals.
3. Instrucțiunea SELECT se va executa cu succes : adevărat / fals.
SELECT last_name, job_id, salary SALARY FROM EMPLOYEES;
4. Instrucțiunea SELECT se va executa cu succes : adevărat / fals.
SELECT * FROM salgrade;
5. În instrucțiunea următoare sunt câteva erori. Identificați-le.
SELECT first_name, salary x 12 ANNUAL SALARY FROM EMPLOYEES;
6. Afipați structura tabeliei DEPARTMENTS și conținutul ei.
7. Afipați structura tabeliei EMPLOYEES. Creați o interogare care să afișeze numele angajatului, meseria, data angajării și numărul angajatului. Salvați instrucțiunea într-un fișier abc.sql.
8. Rulați interogarea din fișierul abc.sql.
9. Afipați meseriile distincte din tabela EMPLOYEES.
10. Încărcați fișierul abc.sql în bufferul SQL. Redenumiți numele coloanelor cu: EMPLOYEES#, Job, Hire Date și apoi rulați interogarea.
11. Afipați numele concatenat cu meseria, separate de virgula și un spațiu și numiți coloana EMPLOYEES -JOB and Title.
12. Afipați datele din tabela EMPLOYEES. Separați fiecare coloană cu o virgulă și numiți coloana THE_OUTPUT.
13. Afipați numele și salariul angajaților care câștiga mai mult de 28500\$. Salvați instrucțiunea SQL într-un fișier p1.sql și apoi rulați-l.
14. Afipați numele angajatului cu marca 201 și numărul departamentului în care lucrează.
15. Modificați p1.sql astfel încât să afișați numele și salariul tuturor angajaților a căror salariu nu intră în intervalul 15000\$ și 28500\$. Salvați instrucțiunea în fișierul p3.sql și apoi rulați-l.
16. Afipați numele, meseria și data de început pentru cei care s-au angajat între 20.02.1985 și 1.05.1992. Afisarea va fi făcută în ordinea crescătoare a datei de angajare.
17. Afipați numele angajaților și numerele de departament ale celor care

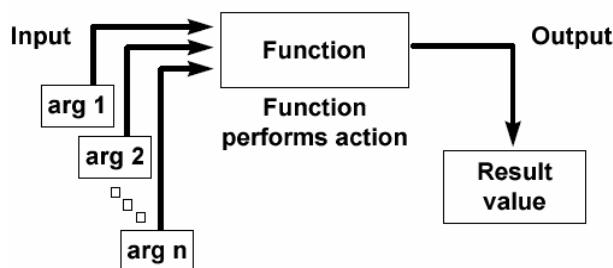
Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Iucrează în departamentele 10 și 30; ordonați alfabetic după nume.

18. Modificați fișierul p3.sql și afișați numele și salariul celor care câștiga mai mult de 15000\$ și sunt în departamentele 10 sau 30. Redenumiți coloanele Angajat și Salar Lunar. Salvați modificările în fișierul p6.sql și apoi rulați-l.
19. Afișați numele și data angajării pentru cei care au fost angajați în anul 1995.
20. Afișați numele și meseria pentru angajații care nu au manager.
21. Afișați numele, salariul și comisionul pentru toți angajații care au comision. Sortați datele în ordine descrescătoare după salariu și comision.
22. Afișați numele angajaților care au a treia literă a numelui 'A'.
23. Afișați numele angajaților care au 2 de 'L' oriunde în numele lor și sunt din departamentul 30 împreuna cu acei angajați care au managerul cu marca 100.
24. Afișați numele, meseria și salariul pentru toți cei care sunt funcționari sau analiști iar salariul lor nu este egal cu 10000\$ sau 30000\$ sau 50000\$.
25. Modificați p6.sql și afișați numele, salariul și comisionul pentru toți angajații care au comisionul mai mare decât salariul mărit cu 10%. Salvați modificările în fișierul p13.sql și apoi rulați-l.

Capitolul 7 Funcții de un singur rând

Funcțiile fac blocul de baza al interogării mai puternic și sunt folosite pentru a manipula date. Acesta capitol este primul dintr-un set de două capitole ce au ca obiectiv descrierea funcțiilor. Ea se ocupă atât de funcțiile de un singur rând pentru caractere, numere și date calendaristice cât și de funcțiile ce fac conversii dintr-un tip de date în altul – de exemplu: din caracter în număr.



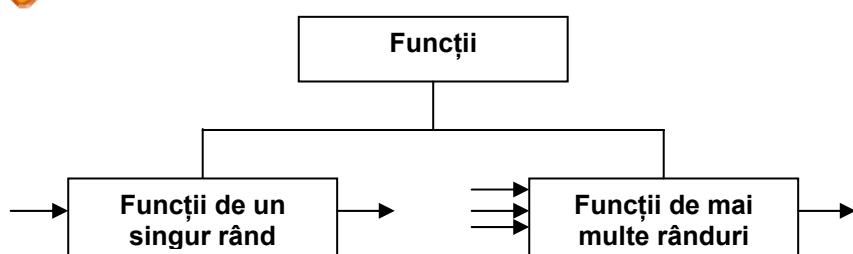
Funcțiile reprezintă o componentă importantă a limbajului SQL și pot fi utilizate pentru :

- Calcule matematice asupra datelor;
- Modificarea unor articole individuale;
- Manipularea ieșirii pentru grupuri de rânduri;
- Stabilirea unui format pentru date calendaristice și numere atunci când acestea sunt tipărite pe ecran;
- Schimbarea tipului de date a unei coloane.

Funcțiile SQL acceptă argumente și întorc valori.



NOTĂ : Majoritatea funcțiilor descrise în acest capitol sunt specifice versiunii SQL pentru Oracle.



Există două tipuri distințe de funcții:

- Funcții de un singur rând
- Funcții de mai multe rânduri

Funcții de un singur rând

Aceste funcții acționează doar asupra unui singur rând și întorc un rezultat pentru fiecare rând. Există mai multe tipuri de funcții de un singur rând. Acest capitol se ocupă de următoarele tipuri: Caracter, Număr, Data calendaristică, Conversie.

Funcții de mai multe rânduri

Aceste funcții acționează asupra unor grupuri de rânduri și întorc un rezultat pentru fiecare grup.

Pentru mai multe detalii consultați lucrarea: Oracle Server SQL Reference. Release 9i pentru o listă completă a funcțiilor disponibile împreună cu sintaxa aferentă.

7.1 Funcții de un singur rând

Funcțiile de un singur:

- Manipulează articole;
- Acționează asupra fiecărui rând rezultat din interogare ;
- Întorc un singur rezultat pentru fiecare rând;
- Pot modifica tipuri de date;
- Pot fi imbricate.

unde:



```
nume_funcție (coloana | expresie,  
[arg1, arg2, ...] )
```

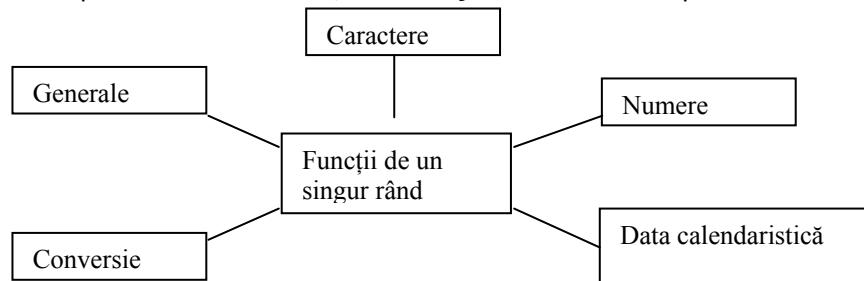
nume_funcție este numele funcției
coloana este un nume de coloană din baza de date
expresie este orice sir de caractere sau expresie calculabilă
arg1, arg2,... sunt argumentele utilizate de funcție

Funcțiile de un singur rând sunt utilizate pentru a manipula date. Ele acceptă unul sau mai multe argumente și întorc o singură valoare pentru fiecare rând rezultat din interogare. O funcție poate avea ca argument unul din următoarele:

- o constantă furnizată de utilizator;
- o variabilă;
- o denumire de coloană;
- o expresie.

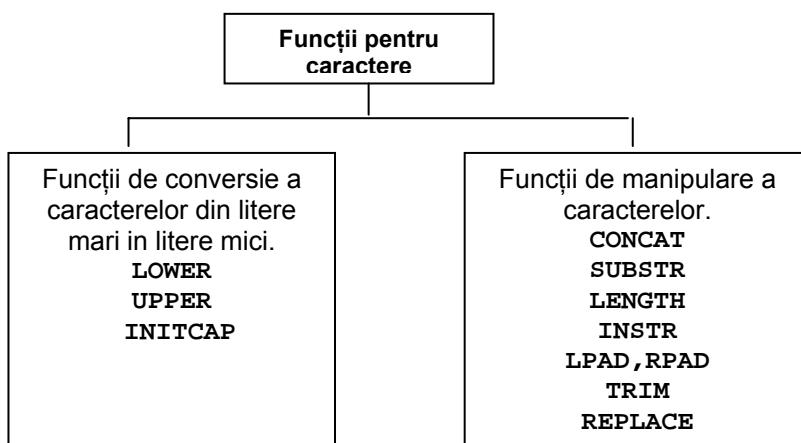
Caracteristici ale funcțiilor de un singur rând

- Acționează asupra fiecărui rând întors de interogare;
- Întorc o valoare pentru fiecare rând;
- Pot întoarce o dată a cărui tip este diferit de tipul argumentului;
- Este posibil să aștepte unul sau mai multe argumente;
- Se pot utiliza în **SELECT, WHERE și ORDER BY**. Se pot imbrica.



Acest capitol prezintă următoarele tipuri de funcții:

- **Funcții pentru caractere**: acceptă argumente de tip caracter și întorc rezultate de tip caracter sau numeric;
- **Funcții pentru numere**: acceptă argumente de tip numeric și întorc rezultate de tip numeric;
- **Funcții pentru date calendaristice**: acceptă argumente de tip dată calendaristică și întorc rezultate de tip dată calendaristică cu excepția funcției **MONTH_BETWEEN** care întoarce o valoare numerică;
- **Funcții pentru conversie**: fac conversia dintr-un tip de data în altul;
- **Funcții generale**: **NVL,NVL2,NULLIF,COALSECE,CASE,DECODE**.



7.2 Funcții pentru caractere

Funcțiile de un singur rând pentru caractere acceptă argumente de tip caracter și întorc rezultate de tip caracter sau numeric. Funcțiile pentru caractere se pot împărți în :

- Funcții de conversie a caracterelor din litere mari în litere mici.
- Funcții de manipulare a caracterelor

Funcție	Scop
LOWER (expresie coloană)	Face conversia caracterelor alfabetice în litere mici
UPPER (expresie coloană)	Face conversia caracterelor alfabetice în litere mari
INITCAP (expresie coloană)	Face conversia pentru primul caracter din fiecare cuvânt în litera mare iar pentru restul caracterelor conversia se face în litere mici
CONCAT(expresie coloana1, expresie coloana2)	Concatenează prima valoare de tip caracter cu a doua valoare de tip caracter. Aceasta funcție este echivalentă cu operatorul de concatenare ().
SUBSTR(expresie coloana, m,/n/)	Întoarce un sir de caractere din cadrul valorii de tip caracter începând cu poziția m și având lungimea n. Dacă m este negativ atunci poziția de început a numărării se consideră a fi ultimul caracter din sir. Dacă n este omis atunci funcția întoarce toate caracterele de la poziția m până la sfârșitul sirului.
LENGTH(expresie coloana)	Întoarce numărul de caractere dintr-o valoare de tip caracter.
INSTR(expresie coloana,m)	Întoarce poziția în cadrul valorii de tip caracter a caracterului specificat.
LPAD (expresie coloana, n,' sir caractere')	Aliniază valoarea de tip caracter la dreapta pe o lungime de n caractere.
TRIM LENGTH(leading trailing both, trim_character, trim_source)	Permite eliminarea caracterelor de la începutul, sfârșitul, ambelor. Trim_character este un literal, trebuie inclusa între apostrof.
REPLACE (text, search_string, replacement_string)	Caută în search_string sirul de caractere specificat prin text, și dacă îl găsește îl înlocuiește cu valoarea din replacement_string.



Funcție	Rezultat
LOWER ('SQL Course')	sql course
UPPER ('SQL Course')	SQL COURSE
INITCAP ('SQL Course')	Sql Course

Functii de conversie a caracterelor din litere mari în litere mici.

Cele trei funcții de conversie a caracterelor sunt: **LOWER, UPPER, INITCAP**.

LOWER: Face conversia în litere mici pentru un text scris cu litere mari și mici.

UPPER : Face conversia în litere mari pentru un text scris cu litere mari și mici.

INITCAP : Face conversia pentru prima literă din fiecare cuvânt în literă mare iar pentru celelalte litere ale cuvântului conversia se face în literă mică.



```
SELECT 'The job id for ' || UPPER(last_name) || ' is ' ||
LOWER(job_id) AS "EMPLOYEE DETAILS" FROM employees;
```

EMPLOYEE DETAILS
The job id for KING is ad_pres
The job id for KOCHHAR is ad_vp
The job id for DE HAAN is ad_vp
The job id for HUNOLD is it_prog
The job id for ERNST is it_prog
... ... is ac_mgr
The job id for GIETZ is ac_account

20 rows selected.



Afişați numărul de ordine, numele și departamentul la care lucrează pentru angajatul Higgins.

```
SELECT employee_id, last_name, department_id
FROM employees WHERE last_name='higgins';
```

No rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

```
SELECT employee_id, last_name, department_id  
FROM employees  
WHERE LOWER(last_name)='higgins';
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
205	Higgins	110

Clauza WHERE din prima instrucțiune SQL specifică numele angajatului ca fiind higgins. Din moment ce toate informațiile din tabela EMPLOYEES sunt memorate ca fiind scrise cu capitalizare, numele 'higgins' (scris cu litere mici) nu poate fi găsit și ca urmare nu se afișează nimic.

Clauza WHERE din cea de-a doua instrucțiune SQL face mai întâi conversia numelui memorat în tabela în litere mici și compară rezultatul obținut cu numele 'higgins'. În acest caz ambii termeni din comparație sunt scriși cu litere mici și deci, de aceasta dată, se pot selecta informațiile necesare din tabelă. Clauza WHERE mai poate fi scrisă ca în exemplul de mai jos, efectul instrucțiunii fiind același. ... **WHERE (last_name)=‘Higgins’**

Numele angajatului din partea dreapta a comparației este scris cu capitalizare adică aşa cum apare în tabela. Pentru a afișa numele cu prima literă convertită în litera mare iar restul în litere mici utilizați funcția **INITCAP**.

Functii pentru manipulat caractere



FUNCȚIE	REZULTAT
CONCAT ('Good','String')	GoodString
SUBSTR ('String',1,3)	Str
LENGTH ('String')	6
INSTR ('String','r')	3
LPAD (salary,10,'*')	*****5000

Functiile pentru manipulat caractere prezentate în cadrul acestei lecții sunt: **CONCAT, SUBSTR, LENGTH, INSTR și LPAD**.

CONCAT Concatenează cei doi parametri. Funcția limitează numărul parametrilor la 2.

SUBSTR Extragă un sir de caracter de o lungime specificată.

LENGTH Întoarce lungimea sirului de caractere (întoarce o valoare numerică).

INSTR Găsește poziția caracterului specificat.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LPAD

Întoarce un sir de caractere rezultat prin inserarea valorii specificate în cel de-al treilea argument la stânga primului argument lungimea rezultatului având lungimea specificată de cel de-al doilea argument.

RPAD

Are un comportament similar cu funcția **LPAD** numai că inserarea celui de-al treilea argument se face la dreapta primului argument.



```
SELECT employee_id,CONCAT(first_name,last_name) NAME,
Job_id, LENGTH (last_name),
INSTR(last_name,'a')"Contains 'a'? "
FROM employees
WHERE SUBSTR(job_id,4)='REP';
```

EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
174	EllenAbel	SA_REP	4	0
176	JonathonTaylor	SA_REP	6	2
178	KimberelyGrant	SA_REP	5	3
202	PaiFay	MK_REP	3	2

Exemplul de mai sus afișează numele angajatului și funcția sa împreuna, lungimea numelui și poziția literei a în cadrul numelui, pentru toate persoanele care au funcția de vânzător.

Modificați exemplul de mai sus astfel încât instrucțiunea SQL să afișeze informațiile despre angajați pentru acele persoane a căror nume se termină în litera n.



```
SELECT employee_id,CONCAT(first_name,last_name) NAME,
Job_id, LENGTH (last_name),
INSTR(last_name,'a')"Contains 'a'? "
FROM employees WHERE SUBSTR(last_name,-1,1)='n';
```

EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
102	LexDe Haan	7	5
200	JenniferWhalen	6	3
201	MichaelHartstein	9	2

7.3 Funcții pentru valori numerice

ROUND Rotunjește valoarea cu un număr specificat de zecimale.
 $\text{ROUND}(45.926,2) \rightarrow 45.93$

TRUNC Trunchiază valoarea
 $\text{TRUNC}(45.926,2) \rightarrow 45.92$

MOD Întoarce restul împărțirii
 $\text{MOD}(1600,300) \rightarrow 100$

Funcțiile pentru valori numerice acceptă valori numerice și întorc valori numerice. Aceasta secțiune descrie o parte din aceste funcții:

Funcție	Scop
ROUND (coloana expresie, n)	Rotunjește coloana, expresia sau valoarea la un număr cu n poziții la partea zecimală. Dacă n este omis numărul rezultat din conversie nu are parte zecimală. Dacă n este negativ este rotunjit numărul din partea stânga a punctului zecimal.
TRUNC (coloana expresie, n)	Trunchiază coloana, expresia sau valoarea la un număr cu n poziții la partea zecimală. Dacă n este omis numărul rezultat din conversie nu are parte zecimală. Dacă n este negativ este trunchiat numărul din partea stânga a punctului zecimal către zero.
MOD (m,n)	Întoarce restul împărțirii dintre m și n.

Utilizarea funcției ROUND



**SELECT ROUND(45.923,2), ROUND(45.923,0),ROUND(45.923,-1)
FROM DUAL;**

ROUND(45.923,2) ROUND(45.923,0) ROUND(45.923,-1)

45.92 46 50

Funcția **ROUND** rotunjește coloana, expresia sau valoarea la un număr cu n poziții la partea zecimală.

- Dacă al doilea argument este omis sau este 0 numărul rezultat din conversie nu are parte zecimală.
- Dacă al doilea argument este 2 atunci numărul rezultat din conversie are 2 cifre la partea zecimală.
- Dacă al doilea argument este -2 atunci se rotunjesc primele 2 cifre ale numărului de la stânga punctului zecimal.



Funcția **ROUND** poate fi utilizată asupra datelor calendaristice.



NOTĂ: DUAL este o tabelă fictivă. Mai multe detalii despre acest aspect vor fi oferite mai târziu.

Utilizarea funcției TRUNC



```
SELECT TRUNC(45.923,2), TRUNC(45.923),TRUNC(45.923,-1)  
FROM DUAL;
```

TRUNC(45.923,2) TRUNC(45.923) TRUNC(45.923,-1)

45.92 45 40

Funcția **TRUNC** trunchiază coloana, expresia sau valoarea la un număr cu n poziții la partea zecimală.

Funcția **TRUNC** funcționează ca și funcția **ROUND**.

- Dacă al doilea argument este omis sau este 0 numărul rezultat din conversie nu are parte zecimală.
- Dacă al doilea argument este 2 atunci numărul rezultat din conversie are 2 cifre la partea zecimală.

Funcția **TRUNC**, la fel ca și funcția **ROUND**, poate fi utilizată asupra datelor calendaristice.

Utilizarea funcției MOD



```
SELECT last_name, salary, MOD(salary, 5000)  
FROM employees  
WHERE job_id = 'SA_REP';
```

LAST_NAME	SALARY	MOD(SALARY,5000)
Abel	11000	1000
Taylor	8600	3600
Grant	7000	2000

Funcția **MOD** întoarce restul împărțirii dintre valoarea1 și valoarea2. Exemplul de mai sus calculează restul împărțirii dintre salar și 5000 pentru toți angajații care sunt agenți comerciali.

7.4 Utilizarea datelor calendaristice

- Oracle memorează datele calendaristice într-un format numeric intern: Secol, an, luna, zi, ora, minute, secunde.
- Formatul implicit pentru date calendaristice este: DD-MON-YY.
- **SYSDATE** este o funcție care întoarce data și timpul.
- **DUAL** este o tabelă fictivă utilizată pentru a vedea rezultatul întors de anumite funcții precum **SYSDATE**.

Formatul datei calendaristice în Oracle

Oracle memorează datele calendaristice într-un format numeric intern: Secol, an, luna, zi, ora, minute, secunde. Formatul implicit pentru date calendaristice este: DD-MON-YY. Valorile valide pentru date calendaristice se situează între Ianuarie 1. 4712 B.C. și Decembrie 31. 9999 A.D.

SYSDATE este funcția care întoarce data și timpul curent. **SYSDATE** se poate utiliza identic cu orice denumire de coloană. De exemplu se poate afișa data curentă selectând **SYSDATE** dintr-o tabelă. Tabela pe care o folosiți rămâne la latitudinea dumneavoastră. Se poate folosi pentru afișarea datei tabela **DUAL**.

Tabela **DUAL** este proprietatea utilizatorului SYS și poate fi accesată de toți utilizatorii. Ea conține o coloană **DUMMY**, și un rând cu valoarea X. Tabela **DUAL** este folositoare atunci când avem de întors o singură valoare ca de exemplul valoare unei constante, pseudocoloane sau o expresie care nu este derivată dintr-o tabela cu date utilizator.

 **Exemplu:** afișarea datei curente folosind tabela **DUAL**.

SELECT SYSDATE FROM DUAL;

Operații aritmetice cu date calendaristice

- **Dacă adunați sau scădeți un număr la sau dintr-o dată calendaristică veți obține tot o dată calendaristică.**
- **Scădeți două date pentru a găsi numărul de zile dintre acestea.**

Din moment ce baza de date memorează datele calendaristice ca numere rezultă că asupra acestor date se pot efectua operații aritmetice utilizând operatori aritmetici cum ar fi + și -. Deasemeni se pot aduna sau scădea constante numerice la date calendaristice. Există posibilitatea efectuării următoarelor operații:

Operatie	Rezultat	Descriere
data + număr	dată	adună un număr de zile la o dată
data – număr	dată	scade un număr de zile dintr-o dată
data – data	număr de zile	scade o dată din cealaltă
data + număr/24	dată	adună un număr de ore la o dată



```
SELECT last_name, (SYSDATE-hire_date)/7 WEEKS
FROM employees
WHERE department_id = 90;
```

LAST_NAME	WEEKS
King	716.227563
Kochhar	598.084706
De Haan	425.227563

Exemplul de mai sus prezintă o tabelă cu numele angajaților din departamentul 90 și de perioada în care au fost angajați exprimată în săptămâni. Pentru a afișa perioada angajării în săptămâni se face diferența între data curentă (data **SYSDATE**) și data la care a fost angajată persoana și apoi se împarte rezultatul la 7.



NOTĂ: **SYSDATE** este o funcție SQL ce întoarce data și timpul curent. Rezultatul pe care îl obțineți dacă probați exemplul poate să difere de rezultatul de mai sus.

7.5 Funcții pentru date calendaristice

Funcție	Descriere
MONTHS_BETWEEN	Întoarce numărul de luni dintre două date calendaristice.
ADD_MONTHS	Adună un număr de săptămâni la o dată calendaristică.
NEXT_DAY	Întoarce ziua ce urmează datei specificate
LAST_DAY	Ultima zi a lunii.
ROUND	Rotungește data calendaristică.
TRUNC	Trunchiază data calendaristică.

Facultatea de Automatică și Calculatoare Iași ***Baze de date – lucrări practice***

Funcțiile pentru date calendaristice operează asupra datelor calendaristice de tip Oracle. Toate funcțiile pentru date întorc o valoare de tip dată cu excepția funcției **MONTH_BETWEEN**, care întoarce o valoare numerică.

MONTHS_BETWEEN (data1,data2): Găsește numărul de luni dintre data1 și data2. Rezultatul poate fi pozitiv sau negativ.

- Dacă data1 este mai târzie decât data2 atunci rezultatul este pozitiv.
- Dacă data2 este mai târzie decât data1 atunci rezultatul este negativ. Partea neîntreaga a rezultatului reprezintă o parte din lună.

ADD_MONTHS(data,n): Adună un număr de n luni la *data*. Numărul n trebuie să fie întreg și poate fi negativ.

NEXT_DAY(data,'char'): Determină data calendaristică a următoarei zile specificate, din săptămâna, care urmează datei "data".

LAST_DAY(data): Determină data calendaristică a ultimei zile specificate, din săptămâna, care urmează datei "data".

ROUND(data[,fmt]): Întoarce data rotunjită în funcție de formatul fmt. Dacă fmt este omis atunci data este rotunjită la cea mai apropiată dată.

TRUNC(data[,fmt]): Întoarce data "data" trunchiată în funcție de formatul fmt. Dacă fmt este omis atunci data este rotunjită la cea mai apropiată zi.

Aceasta listă este un subset al funcțiilor disponibile. Modelele pentru format sunt tratate mai târziu în cadrul acestui capitol. Exemplele de format sunt: month și year.



MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	→ 19.6774194
ADD_MONTHS ('11-JAN-94',6)	→ '11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	→ '08-SEP-95'
LAST_DAY ('01-SEP-95')	→ '30-SEP-95'

Pentru toate persoanele care au fost angajate pe o perioadă mai mică de 200 de luni, afișați numărul de ordine al angajatului, data angajării, numărul de luni pe care persoana le-a acumulat ca angajat, data reviziei care trebuie făcută peste 6 luni, prima vineri de după data angajării, ultima zi a lunii în care s-a făcut angajarea.



```
SELECT employee_id, hire_date,  
MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,  
ADD_MONTHS (hire_date, 6) REVIEW,  
NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)  
FROM employees  
WHERE MONTHS_BETWEEN (SYSDATE, hire_date) < 36;
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY()	LAST_DAY()
107	07-FEB-99	25.0548529	07-AUG-99	12-FEB-99	28-FEB-99
124	16-NOV-99	15.7645303	16-MAY-00	19-NOV-99	30-NOV-99
143	15-MAR-98	35.7967884	15-SEP-98	20-MAR-98	31-MAR-98
144	09-JUL-98	31.9903368	09-JAN-99	10-JUL-98	31-JUL-98
149	29-JAN-00	13.3451755	29-JUL-00	04-FEB-00	31-JAN-00
176	24-MAR-98	35.5064658	24-SEP-98	27-MAR-98	31-MAR-98
178	24-MAY-99	21.5064658	24-NOV-99	28-MAY-99	31-MAY-99

7 rows selected.

Funcțiile **ROUND** și **TRUNC** pot fi utilizate atât pentru numere cât și pentru date calendaristice. Atunci când sunt utilizate cu date calendaristice, acestea rotunjesc sau trunchiază data ținând cont de modelul specificat. Astfel se pot rotunji, de exemplu, date calendaristice spre cel mai apropiat an sau cea mai apropiată lună.



ROUND ('25-JUL-95', 'MONTH')	→ 01-AUG-95
ROUND ('25-JUL-95', 'YEAR')	→ 01-JAN-96
TRUNC ('25-JUL-95', 'MONTH')	→ 01-JUL-95
TRUNC ('25-JUL-95', 'YEAR')	→ 01-JAN-95

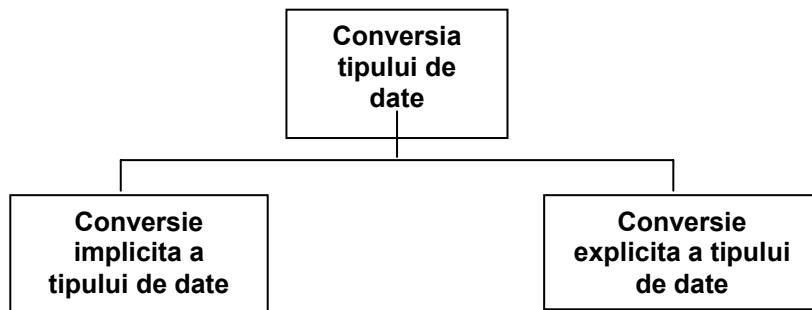
Exemplu:

Comparați datele în care s-au făcut angajări pentru toate persoanele care au început să lucreze în anul 1997. Afipați numărul de ordine al angajatului, data angajării, și luna în care acesta a început să lucreze exprimată sub forma unui interval, folosind funcțiile **ROUND** și **TRUNC**.



```
SELECT employee_id, hire_date,
ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
FROM employees WHERE hire_date like '%97' ;
```

7.6 Funcții pentru conversia tipului de date



Pe lângă tipurile de date din Oracle, coloanele tabelelor dintr-o baza de date Oracle pot fi definite utilizând tipuri de date ANSI, DB2 și SQL/DS. Intern server-ul Oracle face conversia din aceste tipuri de date în tipuri de date Oracle.

În unele situații, server-ul Oracle acceptă anumite tipuri de date deși în mod normal ar trebui să primească alte tipuri. Acest lucru se întâmplă atunci când server-ul Oracle poate face automat conversia în tipul de date pe care îl aşteaptă. Aceste conversii se pot face *implicit* efectuate de către server-ul Oracle sau *explicit* efectuate de către utilizator.

Conversiile de date implicite se fac conform unui set de reguli ce va fi detaliat mai târziu.

Conversiile de date explicite se fac utilizând funcții de conversie. Funcțiile de conversie transformă tipul unei valori în altul. În general funcțiile de conversie respectă următoarea formă:

tip de data1 TO tip de data2,

unde:

tip de data1 este tipul de data care trebuie transformat și reprezintă intrarea, iar

tip de data2 este tipul de data spre care se face conversia și reprezintă ieșirea.



NOTĂ: Deși se fac conversii de date în mod implicit atunci când este nevoie, este recomandat ca aceste conversii să fie făcute implicit de către utilizator pentru a asigura corectitudinea instrucțiunilor.

Conversii de date implicate

În operații de atribuire Oracle poate automat conversia:

DIN	IN
VARCHAR2 sau CHAR (șir de caractere)	NUMBER (valoare numerică)
VARCHAR2 sau CHAR (șir de caractere)	DATE
NUMBER (valoare numerică)	VARCHAR2 (șir de caractere)
DATE (data calendaristică)	VARCHAR2 (șir de caractere)

Operația de atribuire are loc cu succes dacă server-ul Oracle poate converti tipul de dată al sursei în tipul de dată al destinației.

În cazul evaluării expresiilor, Oracle poate automat conversia:

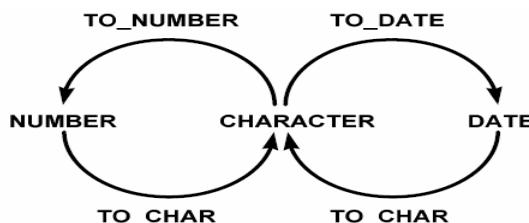
DIN	IN
VARCHAR2 sau CHAR (șir de caractere)	NUMBER (valoare numerică)
VARCHAR2 sau CHAR (șir de caractere)	DATE (data calendaristică)

În general server-ul Oracle utilizează regulile de conversie pentru expresii în cazul în care regulile de conversie pentru atribuire nu acoperă și situația respectivă.



NOTĂ: Conversia din CHAR în NUMBER are loc cu succes doar dacă șirul de caractere reprezintă un număr valid. Conversia din CHAR în DATE are loc cu succes doar dacă șirul de caractere respectă formatul implicit: DD-MON-YY.

Conversii de date explicite



SQL pune la dispoziție trei funcții cu ajutorul cărora se pot face conversii dintr-un tip de date în altul.

Funcție	Scop
TO_CHAR (număr data calendaristică, ['fmt'])	Face conversia dintr-un număr sau o dată calendaristică într-un sir de caractere de tipul VARCHAR2 respectând formatul fmt specificat.
TO_NUMBER (caracter)	Face conversia dintr-un sir de caractere ce conține cifre într-o valoare numerică.
TO_DATE (caracter ,['fmt'])	Face conversia dint-un sir de caractere ce reprezintă o data într-o valoare de tip DATE respectând formatul fmt specificat. (Dacă fmt este omis, formatul implicit este DD-MON-YY)

NOTĂ: Lista prezentată mai sus reprezintă un subset din funcțiile disponibile pentru conversii. Pentru mai multe detalii consultați lucrarea: Oracle Server SQL Reference. Release 9i “Conversion Function”



Utilizarea funcției TO_CHAR împreună cu date calendaristice



TO_CHAR (data calendaristica, 'fmt')

Modelul de formatare:

Trebuie inclus între ghilimele simple și este:

- **case sensitive;**
- **Poate include orice element valid al modelului de formatare pentru date calendaristice ;**
- **Are un element *fm* care elibera spațiile albe sau zerourile nesemnificative;**
- **Este separat de data calendaristică prin virgulă;**
- **Determină afișarea datei calendaristice într-un anumit format.**

Până acum toate datele calendaristice au fost afișate respectând formatul **DD-MON-YY**. Funcția **TO_CHAR** permite conversia din formatul implicit într-un format specificat de dumneavoastră.



Observații:

- Modelul de format trebuie inclus între ghilimele simple și este case sensitive;
- Modelul de format poate include orice element valid al modelului de formatare pentru date calendaristice. Asigurați-vă că valoarea este separată de modelul de formatare prin virgulă;
- Pentru numele zilelor și al lunilor se adăuga automat spații goale la ieșire ;
- Pentru a elimina spațiile și zerourile nesemnificative folosiți elementul pentru modul de umplere;
- Aveți posibilitatea de a redimensiona lungimea pe care se face afișarea pentru un câmp cu ajutorul comenzi SQL*Plus COLUMN.
- Lungimea implicită a coloanei rezultate este de 80 caractere.



```
SELECT employee_id, TO_CHAR (hire_date, 'MM/YY')
Month_Hired
FROM emp
WHERE ename='Higgins';
```

Elementele ale modelului de formatare pentru date calendaristice

YYYY	Anul afișat pe 4 cifre
YEAR	Anul în litere
MM	Luna scrisă cu două cifre
MONTH	Numele lunii
DY	O abreviație a denumirii unei zile din săptămâna formată din trei litere
DAY	Denumirea completa a zilei

Exemple de elementele ale modelului de formatare:

Element	Descriere
SCC sau CC	Secolul
YYYY sau SYYYY	Anul
YYY sau YY sau Y	Ultimile 3, 2 sau 1 cifre din an
Y,YYY	O virgula în cadrul anului

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

[YYY,[YY,[Y,	4,3,2 sau o cifră din an conform standardului ISO
SYEAR sau YEAR	Anul în litere
BC sau AD	Indicatorul B.C. sau A.D.
B.C. sau A.D.	Indicatorul B.C. sau A.D.
Q	Sfertul unui an
MM	Luna scrisă cu două cifre
MONTH	Numele întreg al lunii scris pe 9 caractere. Dacă denumirea lunii nu ocupă cele 9 caractere, spațiul rămas liber este automat umplut cu spații
MON	O abreviație a denumirii unei luni formată din trei litere
RM	Luna scrisă cu cifre romane
WW sau W	Săptămâna din an sau luna
DDD sau DD sau D	Ziua din an, lună sau săptămână.
DAY	Denumirea completă a zilei completată eventual cu spații până la 9 caractere.
DY	O abreviație a denumirii unei zile formată din trei litere
J	Numărul de zile de la data de 31 Decembrie 4713BC

Elemente ce formătăză timpul

HH24:MI:SS AM	15:45:32 PM
----------------------	--------------------

Adăugați siruri de caractere prin închiderea acestora între ghilimele.

DD “of” MONTH	12 of OCTOBER
----------------------	----------------------

Adăugați sufixe pentru a scrie în litere un număr .

ddspth	fourteenth
---------------	-------------------

Modele de formatare pentru timp

Utilizați elementele descrise mai jos atunci când doriți să afișați timpul într-un anumit format sau să folosiți litere în loc de cifre.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Element	Descriere
AM sau PM	indicator de meridian
A.M. sau P.M.	indicator de meridian cu puncte
HH sau HH12 sau HH24	ora
MI	minute (0-59)
SS	secunde (0-59)
SSSS	Numărul de secunde începând cu miezul nopții

Alte formate:

Element	Descriere
/..	Punctuația este reprodusă în rezultat.
“of the”	Şirul încadrat între ghilimele este reprodus

Specificați sufixe :

Element	Descriere
TH	Număr de ordine dat în cifre (de exemplu DDTH pentru 4TH)
SP	Număr scris în litere (de exemplu DDSP pentru FOUR)
SPTH sau THSP	Număr de ordine scris în litere (de exemplu DDSPTH pentru FOURTH)

Utilizarea funcției TO_CHAR împreună cu date calendaristice



```
SELECT last_name,TO_CHAR(hire_date, 'fmDD Month YYYY')
HIRE_DATE FROM employees;
```

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

LAST_NAME	HIRE_DATE
King	17 June 1987
Kochhar	21 September 1989
De Haan	13 January 1993
Hunold	3 January 1990
Ernst	21 May 1991
Austin	25 June 1997
Pataballa	5 February 1998
Lorentz	7 February 1999
Greenberg	17 August 1994
Faviet	16 August 1994
Chen	28 September 1997

Exemplul de mai sus prezintă o modalitate de a afișa numele și data angajării pentru fiecare angajat.(De remarcat este formatul în care se afișează data.)



Modificați exemplul de mai sus astfel încât data calendaristică sa aibă următorul format: Seventh of February 1981 08:00:00 AM.

**SELECT last_name,TO_CHAR (hire_date, 'fmDdspth "of" Month
YYYY fmHH:MI:SS AM') HIRE_DATE FROM employees;**

LAST_NAME	HIRE_DATE
King	Seventeenth of June 1987 12:00:00 AM
Kochhar	Twenty-First of September 1989 12:00:00 AM
De Haan	Thirteenth of January 1993 12:00:00 AM
Hunold	Third of January 1990 12:00:00 AM
Ernst	Twenty-First of May 1991 12:00:00 AM
Austin	Twenty-Fifth of June 1997 12:00:00 AM
Pataballa	Fifth of February 1998 12:00:00 AM
Lorentz	Seventh of February 1999 12:00:00 AM
Greenberg	Seventeenth of August 1994 12:00:00 AM
Faviet	Sixteenth of August 1994 12:00:00 AM
Chen	Twenty-Eighth of September 1997 12:00:00 AM

De remarcat este faptul că denumirea lunii respectă modelul pentru format specificat (INITCAP).

Utilizarea funcției TO_CHAR împreuna cu valori numerice



TO_CHAR (number, 'fmp')

Pentru a afișa o valoare numerică sub forma unui caracter utilizați următoarele elemente de formatare împreuna cu funcția **TO_CHAR**.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

9	Reprezintă un număr
0	Forțează afișarea unei cifre 0
\$	Semnul dolar
L	Folosește simbolul local pentru monedă
.	Afișează un punct
,	Tipărește un indicator pentru mii

Atunci când se lucrează cu valori numerice drept siruri de caractere trebuie să convertiți acele numere în valori de tip caracter utilizând funcția **TO_CHAR**, care face conversia dintre o valoare de tip **NUMBER** spre o valoare de tip **VARCHAR2**. Aceasta tehnică este folositoare în cadrul unei concatenări.

Elemente de formatare pentru numere

Dacă trebuie să convertiți un număr într-o valoare de tip caracter puteți utiliza următoarele elemente:

Element	Descriere	Exemplu	Rezultat
9	Pozitie numerică (numărul cifrelor de 9 determină lungimea pe care se face afișarea)	999999	1234
0	Afișează zerourile nesemnificative	099999	001234
\$	Semnul dolar	\$999999	\$1234
L	Folosește simbolul local pentru monedă	L999999	FF1234
.	Determină afișarea unui punct zecimal în poziția specificată.	999999.99	1234.00
,	Determină afișarea unei virgule în poziția specificată.	999,999	1,234
MI	Determină afișarea semnului minus în partea dreaptă (pentru valori negative)	999999MI	1234 -
PR	Închide între paranteze numerele negative	999999PR	<1234>
EEEE	NOTĂ: științifică (formatul impune existența a patru litere E)	99.999EEEE	1.234E+03
V	Înmulțire cu 10 de n ori (n = numărul de cifre de 9 de după litera V)	9999V99	123400
B	Înlocuiește valorile de 0 cu blank	B9999.99	1234.00



```
SELECT TO_CHAR(sal,'$99,999') SALARY  
FROM emp  
WHERE last_name= 'Ernst';
```



Observații:

Server-ul Oracle afișează semnul (#) în locul valorii numerice a cărei număr cifre a depășit valoarea specificată prin model. Server-ul Oracle rotunjește valoarea zecimală stocată ca o valoare cu un număr de zecimale furnizat de către modelul de formatare.

Funcțiile *TO_CHAR* și *TO_DATE*



<i>TO_NUMBER (char)</i>
<i>TO_DATE (char [, 'fmt '])</i>

Pentru a face conversia dintr-un sir de caractere într-un număr folosiți funcția *TO_NUMBER*.

Pentru a face conversia dintr-un sir de caractere într-o dată calendaristică folosiți funcția *TO_DATE*.

Este posibil să apăra o situație în care doriți să faceți conversia dintr-un sir de caractere într-un număr sau într-o dată calendaristică. Pentru a realiza aceste tipuri de conversii utilizați funcțiile *TO_NUMBER* și *TO_DATE*. Modelul după care se face formatarea va trebui alcătuit pe baza elementelor pentru formatare prezentate anterior.

Specificatorul *fx* indică potrivirea perfectă dintre caracterul argument și modelul de formatare a datei.

- Punctuația și textul dintre apostrof trebuie să se potrivească exact cu modelul din clauza de formatare.
- Nu pot să fie scrise blank-uri suplimentare.
- Datele numerice trebuie să aibă același număr de caractere ca și modelul corespunzător din format.



Exemplu: Afișați numele și data angajării pentru toate persoanele care au fost angajate pe 24.05.1999.

```
SELECT last_name, hire_date FROM employees  
WHERE hire_date = TO_DATE ('May 24, 1999', 'fxMonth DD, YYYY');
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	HIRE_DATE
Grant	24-MAY-99

Formatul RR pentru date calendaristice

Anul curent	Data specificata	Formatul RR	Formatul YY
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095
Dacă cele două cifre specificate ale anului sunt:			
0-49		50-99	
Dacă cele două cifre ale anului curent sunt:	0-49	Data întoarsă se încadreză în secolul curent	Data întoarsă se încadreză în secolul de dinaintea celui curent
	50 - 99	Data întoarsă se încadreză în secolul de după secolul curent	Data întoarsă se încadreză în secolul curent

Formatul RR pentru date calendaristice este similar cu elementul YY, dar permite specificarea de secole diferite. Există posibilitatea folosirii elementului pentru formatarea datelor RR în locul elementului YY și astfel secolul valorii returnate variază în funcție de cei doi digiti specificați în an și de ultimii doi digiti ai anului curent. Tabelul următor descrie comportamentul elementului RR.

Anul curent	Data specificata	Formatul RR	Formatul YY
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017

7.7 Funcții diverse

Funcția NVL

- Convertește o valoare nulă într-o valoare efectivă;
- Tipurile de date care pot fi folosite sunt: **dată calendaristică, caracter și număr**;
- Tipurile de date trebuie să se potrivească.
 - **NVL (comm,0)**
 - **NVL (hire_date,'01-JAN-97')**
 - **NVL (job, 'No Job Yet')**



Sintaxa

NVL (expr1, expr2)

unde: expr1 este valoarea sau expresia sursă care ar putea să conțină o valoare nulă.

expr2 este valoarea întâi, valoarea spre care se face conversia

Aveți posibilitatea de a utiliza funcția **NVL** împreună cu orice tip de dată, dar tipul valorii întoarse este de fiecare dată la fel cu tipul parametrului expr1.

Conversii **NVL** pentru diferite tipuri de date

Tip de data	Exemplul de conversie
NUMBER	NVL (coloana ce conține o valoare de tip numeric, 9)
DATE	NVL (coloana ce conține o valoare de tip dată calendaristică, '01-JAN-95')
CHAR sau VARCHAR2	NVL (coloana ce conține o valoare de tip caracter, 'Unavariable')



```
SELECT last_name, salary, NVL(commission_pct,0),
(Salary*12)+(salary*12+NVL (commission_pct,0)) AN_SAL
FROM employees;
```

LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
King	24000	0	576000
Kochhar	17000	0	408000
De Haan	17000	0	408000
Hunold	9000	0	216000
Ernst	6000	0	144000
Austin	4800	0	115200
Pataballa	4800	0	115200
Lorentz	4200	0	100800
Greenberg	12000	0	288000
Faviet	9000	0	216000
Chen	8200	0	196800

Pentru a calcula compensația anuală pentru toți angajații, trebuie să înmulțești salariul lunar cu 12 și apoi să adăugați comisionul.

```
SELECT last_name, salary, NVL(commission_pct,0),
(Salary*12)+(salary*12*commission_pct) AN_SAL FROM employees;
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
Vargas	2500	0	
Russell	14000	.4	235200
Partners	13500	.3	210600
Errazuriz	12000	.3	187200
Cambrault	11000	.3	171600
Zlotkey	10500	.2	151200
Tucker	10000	.3	156000
Bernstein	9500	.25	142500
Hall	9000	.25	135000
Olsen	8000	.2	115200
Cambrault	7500	.2	108000

Din exemplul precedent se poate remarcă faptul că compensația anuală se calculează doar pentru acei angajați care au o valoare pentru comision nenulă. Dacă se întâlnește pe coloana o valoare nulă atunci rezultatul este nul. Pentru a calcula valorile pentru toți angajații trebuie să convertiți valorile nule în valori numerice înainte de a aplica operatorul aritmetic. O soluție corectă pentru o astfel de problemă este prezentată în exemplul precedent celui luat în discuție, exemplul în care pentru conversia valorilor nule s-a folosit funcția **NVL**.

Funcția NVL2



Sintaxa

NVL (expr1, expr2,expr3)

unde: expr1 este valoarea sau expresia sursă care ar putea să conțină o valoare nulă.

expr2 este valoarea returnată dacă expr1 nu este null.

expr3 este valoarea returnată dacă expr1 este null.

Funcția **NVL2** examinează prima expresie.

- Dacă prima expresie nu este null atunci funcția întoarce cea de a doua expresie.
- Dacă prima expresie este null atunci funcția întoarce cea de a treia expresie.



```
SELECT first_name, salary, commission_pct,
NVL2(commission_pct, 'SAL+COMM','SAL') income
FROM employees WHERE department_id IN
(50,80);
```

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Davies	3100		SAL
Matos	2600		SAL
Vargas	2500		SAL
Russell	14000	.4	SAL+COMM
Partners	13500	.3	SAL+COMM
Errazuriz	12000	.3	SAL+COMM
Cambrault	11000	.3	SAL+COMM
Zlotkey	10500	.2	SAL+COMM
Tucker	10000	.3	SAL+COMM
Bernstein	9500	.25	SAL+COMM
Hall	9000	.25	SAL+COMM

În exemplul de mai sus este analizat comisionul. Dacă este sesizată o valoare, se returnează expresia SAL+COMM iar dacă comisionul este null se returnează doar salariul.

Primul argument poate avea orice tip de dată.

Argumentele doi și trei pot fi de orice tip în afară de LONG. Dacă tipurile de date pentru argumentele doi și trei sunt diferite, Oracle convertește automat tipul de date al argumentului trei la cel deținut de argumentul doi înainte de a face comparația, numai dacă argumentul trei nu este null. În acest caz nu este necesară o conversie de date. Tipul de date al expresiei returnate este același cu al argumentului doi numai dacă argumentul doi este de tip caracter, caz în care tipul de date returnat este varchar2.

Funcția **NULLIF**

Funcția **NULLIF** compară două expresii. Dacă ele sunt egale, funcția returnează null, dacă nu, funcția returnează prima expresie. Nu se poate specifica literalul null pentru prima expresie.

Sintaxa

NULLIF (expr1, expr2,)

unde: expr1 este valoarea sau expresia sursă care se compară cu expr2
expr2 este valoarea sau expresia sursă care se compară cu expr1

În exemplu se compară lungimea numelui cu lungimea prenumelui pentru fiecare angajat. În cazul în care acestea sunt egale se returnează null, altfel se returnează lungimea dată de first_name.

```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name, LENGTH(last_name) "expr2",
       NULLIF (LENGTH(first_name), LENGTH(last_name)) result
  FROM employees;
```

FIRST_NAME	expr1 LAST_NAME	expr2	RESULT
Douglas	7 Grant	5	7
Jennifer	8 Whalen	6	8
Michael	7 Hartstein	9	7
Pat	3 Fay	3	
Susan	5 Mavris	6	5
Hermann	7 Baer	4	7
Shelley	7 Higgins	7	
William	7 Gietz	5	7

Funcția **NULLIF** are ca echivalent logic expresia de tip **CASE**.
CASE WHEN expr1=expr2 THEN NULL ELSE expr1 END

Folosirea funcției **COALESCE**

Avantajul folosirii funcției **COALESCE** în locul folosirii funcției **NVL** este faptul că funcția **COALESCE** poate lua mai multe valori alternative.

Dacă prima expresie nu este null, funcția are ca rezultat această expresie altfel, ea unește celelalte expresii.

Funcția **COALESCE** întoarce prima expresie ce nu este null din listă.



SINTAXA

COALESCE (expr1, expr2,... exprn)

unde :

expr1 este întoarsa de funcție dacă expr1 nu are valoarea null
 expr2 este întoarsa de funcție dacă expr1 are valoarea null și expr2 nu are valoarea null
 exprn este întoarsa de funcție dacă precedentele expresii nu au valoarea null



```
SELECT last_name,
COALESCE(commission_pct, salary, 10) comm
FROM employees ORDER BY commission_pct;
```

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

LAST_NAME	COMM
Grant	.15
Zlotkey	.2
Taylor	.2
Acol	.3
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Malos	2000
Vargas	2500

20 rows selected.

Expresia CASE

Expresia de tip **CASE** oferă facilitățile date de o structură de tip **IF THEN ELSE**.



```
CASE expr WHEN comparison_expr1 THEN
    return_expr1
    [WHEN comparison_expr2 THEN return_expr2
    WHEN comparison_expr THEN return_expr
    ELSE else_expr]
END
```

unde: expr1 este valoarea sau expresia sursa care se compară cu expr2
expr2 este valoarea sau expresia sursa care se compară cu expr1

Într-o expresie simplă de tip **CASE** serverul verifică dacă prima condiție de tip **WHEN ... THEN** este egală cu *comparison_expr* și întoarce *return_expr*. Dacă nici una din perechile **WHEN... THEN** nu îndeplinesc condițiile și există o clauză **ELSE** atunci Oracle întoarce *else_expr*. Altfel Oracle întoarce null. Literalul **NULL** nu poate fi specificat în *return_expr* și *else_expr*. Toate expresiile trebuie să fie de același tip de dată.



```
SELECT last_name, job_id,salary,
CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
              WHEN 'ST_CLERK' THEN 1.15*salary
              WHEN 'SA_REP' THEN 1.20*salary
ELSE salary END Revised_salary
FROM employees ;
```

În exemplul de mai sus mărirea de salariu se aplică diferențiat pentru anumite categorii de funcții și anume: programatorilor cu 10%, funcționarilor cu 15% iar vânzătorilor cu 20%. Pentru celelalte funcții salariul rămâne neschimbat.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
King	AD_PRES	24000	24000
Kochhar	AD_UP	17000	17000
De Haan	AD_UP	17000	17000
Hunold	IT_PROG	9000	9900
Ernst	IT_PROG	6000	6600
Austin	IT_PROG	4800	5280
Pataballa	IT_PROG	4800	5280
Lorentz	IT_PROG	4200	4620
Greenberg	FI_MGR	12000	12000
Faviet	FI_ACCOUNT	9000	9000
Chen	FI_ACCOUNT	8200	8200

Funcția DECODE

Facilitează simularea unor structuri de tip CASE sau IF-THEN-ELSE



```
DECODE(col/expression, search1, result1
      [, search2, result2,...,]
      [, default])
```

Funcția **DECODE** evaluează o expresie într-un mod similar structurii IF-THEN-ELSE, structură folosită în multe limbaje de programare. Funcția **DECODE** evaluează expresia după ce o compară cu fiecare valoare *search*. Dacă valoarea expresiei este identică cu valoarea conținută în *search* atunci este întoarsă valoarea *result*.

Dacă valoarea *default* (implicită) este omisă, funcția va întoarce o valoare nulă în cazul în care valoarea expresiei nu este identică cu nici o valoare *search*.



```
SELECT job_id, salary,
       DECODE(job_id, 'IT_PROG', salary*1.1,
              'ST_CLERK', salary *1.15,
              'SA REP', salary *1.20,
              SALARY)REVISED_SALARY
  FROM employees;
```

JOB_ID	SALARY	REVISED_SALARY
AD_PRES	24000	24000
AD_UP	17000	17000
AD_UP	17000	17000
IT_PROG	9000	9900
IT_PROG	6000	6600
IT_PROG	4800	5280
IT_PROG	4800	5280
IT_PROG	4200	4620
FI_MGR	12000	12000
FI_ACCOUNT	9000	9000
FI_ACCOUNT	8200	8200

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

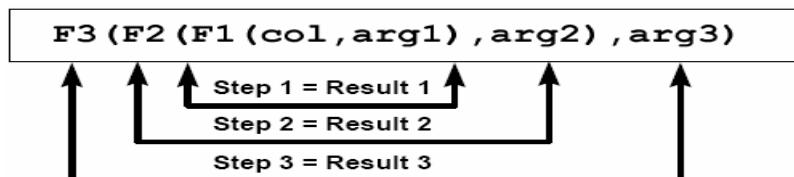
În exemplul de mai sus valoarea evaluată este JOB_ID. Dacă JOB_ID este IT_PROG, sporul de salar este de 10%; Dacă JOB_ID este ST_CLERK, sporul de salar este de 15% iar dacă JOB_ID este SA REP, sporul de salar este de 20%. Pentru celelalte funcții salariile nu se modifică.

Aceeași structura scrisă cu **IF-THEN-ELSE** are următoarea formă:

```
IF job_id = 'IT_PROG' THEN salary = salary * 1.1
IF job_id = 'ST_CLERK' THEN salary = salary * 1.15
IF job_id = 'SA REP' THEN salary = salary * 1.20
ELSE salary = salary
```

7.8 Imbricarea funcțiilor

- Funcțiile de un singur rând se pot imbrica de câte ori dorim.
- Evaluarea funcțiilor imbricate începe de la nivelul cel mai adânc.



Funcțiile de un singur rând se pot imbrica de câte ori dorim. Evaluarea lor se face din centrul expresiei imbricate spre exteriorul acesteia. Exemplele care urmează va vor demonstra flexibilitatea acestor funcții.



```
SELECT last_name, NVL(TO_CHAR(manager_id),'No Manager')
FROM employees WHERE manager_id IS NULL;
```

LAST_NAME	NVL(TO_CHAR(MANAGER_ID), 'NOMANAGER')
King	No Manager

Exemplul de mai sus afișează acele persoane care nu au superior. Evaluarea instrucției SQL se realizează în doi pași.

- Evaluarea funcției din interior ce face conversia dintr-o valoare numerică în una de tip caracter. - Rezultat1=TO_CHAR (mgr)
- Evaluarea funcției din exterior care înlocuiește valorile nule cu un text – NVL (Rezultat1, 'No Manager')

Denumirea coloanei este data de întreaga expresie din moment ce nu este specificat nici un alias pentru acea coloană.



7.9 Exerciții

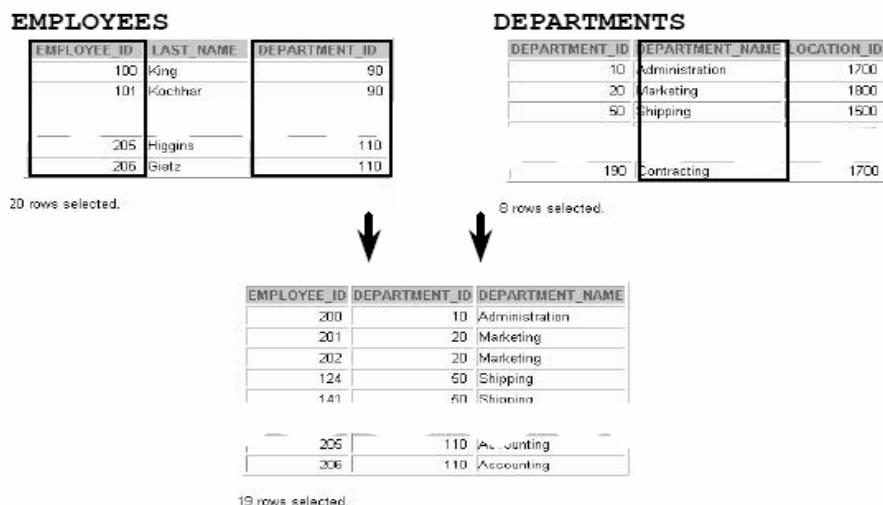
1. Scrieți o interogare care să afișeze data curentă. Denumiți coloana Date.
2. Afișați marca, numele, salariul și salariu mărit cu 15% exprimat ca număr întreg. Denumiți ultima coloana Salar Nou. Salvați instrucțiunea într-un fișier numit p3q2.sql.
3. Rulați programul salvat anterior.
4. Modificați programul salvat în fișierul p3q2.sql astfel încât acesta să adauge o coloana în care veți trece diferența dintre salariul nou și cel vechi. Denumiți coloana Creștere. Rulați noul program.
5. Afișați numele angajatului, data angajării, data când se recalculează salariul, care este prima luni după 6 luni de servici. Denumiți coloana REVIEW. Formatați afișarea datei astfel încât să arate similar cu exemplul de mai jos:
Sunday, the Seventh of September, 1981
6. Pentru fiecare angajat afișați numele și calculați numărul de luni între data de astăzi și data angajării. Denumiți coloana LUNI_DE_ACTIVITATE. Ordonați rezultatul după numărul de luni de lucru. Rotunjiți numărul de luni.
7. Scrieți o interogare care să producă următorul afișaj pentru fiecare angajat:
<nume angajat> câștigă <salariu> lunar dar ar dori <3 * salariu>. Denumiți coloana Salariul de vis.
Exemplu: KING câștiga \$5000 lunar dar ar dori \$15000.
8. Scrieți o interogare care să afișeze numele și salariul pentru toți angajații. Afișați valoarea salariului pe 15 caractere aliniată la dreapta iar spațiul rămas la stânga să fie umplut cu caracterul \$. Denumiți coloana SALARIU.
9. Scrieți o interogare care să afișeze numele angajatului (cu litere mici cu excepția primei litere care se va scrie cu literă mare) și lungimea numelui. Interogarea se face doar pentru angajații a căror nume începe cu una din literele: J, A sau M.
10. Afișați numele, data angajării și ziua din săptămâna în care angajatul a început lucrul. Denumiți coloana ZI. Ordonați rezultatul după cîmpurile coloanei ZI începând cu Monday (Luni).
11. Scrieți o interogare care să afișeze numele angajatului și valoarea comisionului. Dacă angajații nu obțin comision, introduceți "No commission". Denumiți coloana COMM.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Capitolul 8 Afișarea datelor din tabele multiple

Obiective:

- Scrierea expresiilor SELECT pentru a accesa date din mai multe tabele folosind legături (join-uri) de egalitate și inegalitate;
- Vizualizarea datelor care în general nu se acceseză printr-o condiție join simplă ci folosind join-uri exterioare;
- Efectuarea unui join a unui tabel cu el însuși.



Câteodată este necesară afișarea datelor ce provin din mai multe tabele. În exemplul de mai sus rezultatul afișează date din două tabele: **EMPLOYEES** și **DEPARTMENTS**.

- Employee_ID există în tabelul **EMPLOYEES**
- Department_ID există în ambele tabele **EMPLOYEES** și **DEPARTMENTS**
- Location_ID există în tabelul **DEPARTMENTS**

Pentru a se ajunge la rezultatul final tabelele **EMPLOYEES** și **DEPARTMENTS** trebuie să fie legate astfel încât accesarea datelor să se facă din ambele tabele.

8.1 Definirea JOIN-urilor

În vederea afișării de date ce provin din mai multe tabele ale bazei de date se folosește o condiție de join. Liniile dintr-un tabel pot fi alăturate liniilor din alt tabel conform cu valorile comune existente în coloanele corespondente, care sunt de obicei, coloane chei primare și chei străine.

Pentru a afișa date din două sau mai multe tabele aflate în relație se scrie o condiție simplă de tip join în clauza **WHERE**.



```
SELECT    table1.column, table2.column  
FROM      table1, table2  
WHERE     table1.column1 = table2.column2;
```

Unde :

table.column - specifica tabelul și coloana de unde este extrasă data
table1.column1 = table2.column2 - este condiția care alătura (leagă) tabelele împreună

- Condiția de JOIN se scrie în clauza **WHERE**.
- Numele coloanei se prefixează cu numele tabelului când același nume de coloană apare în mai mult de un tabel.



Observații:

- La scrierea unei expresii **SELECT** care leagă tabele precedăti numele coloanei cu numele tabelului pentru claritate și pentru a mari viteza de acces la baza de date.
- Dacă același nume de coloană apare în mai multe tabele numele coloanei trebuie prefixat cu numele tabelului.
- Pentru a alătura n tabele împreună este nevoie de minim $n-1$ condiții de join. Așadar, pentru a alătura 4 tabele, sunt necesare 3 condiții de join. Această regulă s-ar putea să nu se aplique dacă tabelul are o cheie primară concatenată fiind astfel necesare mai multe coloane pentru a identifica în mod unic fiecare linie.

8.2 Produsul Cartezian

Un produs cartezian se formează atunci când:

- o condiție join este omisă;
- o condiție join este invalidă;
- toate liniile din primul tabel sunt alăturate la liniile din tabelul al doilea.

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

Pentru a evita un produs cartezian trebuie să includeți în totdeauna într-o clauza WHERE o condiție de join validă.

Când o condiție join este invalidă sau este complet omisă, rezultatul este un produs cartezian în care vor fi afișate toate combinațiile de linii, adică toate liniile din primul tabel sunt alăturate la toate liniile din cel de-al doilea tabel.

Un produs cartezian tinde să genereze un număr mare de linii și rezultatul sau este rar folositor. În totdeauna într-o clauza **WHERE** trebuie inclusă o condiție de join validă, cu excepția cazului când se dorește în mod explicit combinarea tuturor liniilor din tabele.



Exemplu de generare a unui produs cartezian:

```
SELECT last_name, department_name  
FROM employees, departments;
```

EMPLOYEES (20 rows)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
205	Higgins	110
206	Gietz	110

20 rows selected.

DEPARTMENTS (8 rows)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Shipping	1500
40	IT	1700
50	Customer Service	1700
60	Contracting	1700
70	R&D	1700

8 rows selected.

Cartesian product: →
20x8=160 rows

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	30	Shipping
124	40	IT
141	50	Customer Service
205	60	Contracting
206	70	R&D

160 rows selected.

Un produs cartezian este generat dacă o condiție join este omisă. Exemplul alăturat afișează numele angajatului și numele departamentului din tabelele EMPLOYEES și DEPARTMENTS. Deoarece nu a fost specificată nici o condiție de join în clauza WHERE, toate liniile din tabelul EMPLOYEES sunt alăturate tuturor liniilor din tabelul DEPARTMENTS, generând astfel un tabel de ieșire ce are 160 de rânduri.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Tipuri de Join-uri

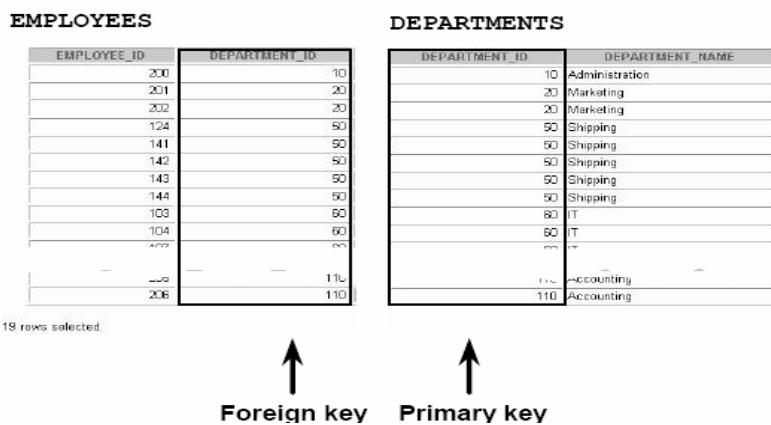
Join –uri specifice Oracle (8i și versiuni anterioare)	Join-uri conforme cu SQL: 1999
Equijoin	Cross Join
Non-equijoin	Natural Join
Outer join	Using Clause
Self join	Full or two-sided outer join
	Arbitrary join conditions for outer join

Oracle 9i pune la dispoziție sintaxa pentru tipuri de join care se conformată cu standardul SQL 1999. Înainte de 9i sintaxa pentru join era diferită de cea cerută de standardele ANSI. Noua sintaxă conformă cu SQL 1999 nu oferă beneficii majore în performanță join-urilor față de cele scrise în sintaxa specifică Oracle înainte de 9i.

8.3 Echi-join

Pentru a determina numele departamentului în care lucrează un angajat trebuie comparate valorile din coloana DEPARTMENT_ID din tabelul EMPLOYEES cu valorile din coloana DEPARTMENT_ID din tabelul DEPARTMENTS. Relația dintre tabelele EMPLOYEES și DEPARTMENTS este un echijoin, aceasta însemnând că valorile din coloana DEPARTMENT_ID din ambele tabele trebuie să fie egale. Acest tip de join-uri implică frecvent existența cheilor primare și străine.

ECHIJOIN -urile sunt adesea numite și **simple join** sau **inner join**.



Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

Extragerea înregistrărilor cu Echijoin-uri



```
SELECT employees.employee_id, employees.last_name,
employees.department_id, departments.department_id,
departments.location_id
FROM employees, departments
WHERE employees.department_id=departments.department_id;
```

În exemplul de mai sus:

- Clauza **SELECT** specifică numele de coloane de extras
 - numele angajatului, numărul angajatului și numărul departamentului care sunt coloane în tabelul EMPLOYEES;
 - numărul departamentului, numele departamentului și locația sunt coloane în tabelul DEPARTMENTS.
- Clauza **FROM** specifică cele 2 tabele din baza de date ce trebuie accesate: tabelul EMPLOYEES și tabelul DEPARTMENTS
- Clauza **WHERE** specifică felul în care tabelele vor fi alăturate.

Deoarece coloana DEPARTMENTS_ID este comună tabelelor ea trebuie prefixată cu numele tabelului pentru a evita ambiguitatea.

Condiții de căutare adiționale folosind operatorul **AND**

În plus față de condiția de join se pot folosi criterii adiționale pentru clauza **WHERE**. De exemplu, pentru a afișa marca angajatului Matos, numele, numărul departamentului și locația departamentului, este nevoie de o condiție suplimentară în clauza **WHERE**.

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
Whalen	10
Hartstein	20
Fay	20
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Hunold	60
Ernst	60
Lorentz	60
Zlotkey	90
Urgo	110
Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
60	IT
60	IT
60	IT
80	Sales
110	Accounting

19 rows selected.



```
SELECT employees.employee_id, employees.last_name,  
departments.location_id  
FROM employees, departments  
WHERE employees.department_id=departments.department_id  
AND Last_name='Matos';
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Matos	50	Shipping

Calificarea numelor de coloane ambigu



- Folosiți prefixele tabelelor pentru a califica numele coloanelor ce sunt comune mai multor tabele;
- Îmbunătățiți performanța prin folosirea prefixelor de tabele;
- Distingeți coloanele care au nume identice în tabele diferite prin folosirea aliasurilor de coloane.

Pentru a evita ambiguitatea trebuie să calificați numele coloanelor în clauza *WHERE* cu numele tabelului. În exemplu, coloana DEPARTMENTS_ID ar putea fi din tabela DEPARTMENTS sau din tabela EMPLOYEES și de aceea este necesară adăugarea prefixului de tabel pentru a executa interogarea.

Dacă cele două tabele nu au nume de coloane comune nu este necesară calificarea coloanelor. Oricum, interogarea va fi mai performantă prin folosirea prefixului de tabel deoarece serverul Oracle î se specifică exact de unde să extragă coloanele.

Cerile de calificare a numelor de coloane ambigu sunt de asemenea aplicabile la coloanele care pot fi ambigu în alte clauze cum ar fi *SELECT* sau *ORDER BY*.

Folosirea de aliasuri pentru tabele

Folosind aliasurile de tabele se simplifică scrierea interogărilor.

Prefixarea numelor coloanelor cu numele tabelului poate consuma mult timp, mai ales dacă numele tabelului este lung. Se pot folosi aliasuri de tabele în locul numelor tabelelor. Așa cum un alias de coloană da unei coloane un alt nume, un alias de tabel îi dă acestuia alt nume. Aliasurile de tabel ajută la scrierea unei linii de cod SQL mai scurte și astfel memoria este mai puțin folosită.



```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM employees e, departments d
WHERE e.department_id=d.department_id;
```

În exemplul de mai sus, în clauza *FROM*, numele tabelului este specificat în întregime fiind urmat de un spațiu și apoi de alias. Tabelului EMPLOYEES i-a fost dat aliasul E, iar tabelului DEPARTMENTS aliasul D.



Observații:

- Aliasurile de tabel pot avea pana la 30 caractere lungime, dar cu cât sunt mai scurte, cu atât mai bine;
- Daca aliasul unui tabel este folosit pentru un nume de tabel particular în clauza *FROM*, atunci acel alias de tabel trebuie sa fie substitutentul pentru numele tabelului pe tot cuprinsul expresiei *SELECT*;
- Aliasurile de tabel trebuie să aibă sens;
- Aliasul de tabel este valid numai pentru *SELECT*-ul curent.

JOIN ce folosește mai mult de 2 tabele

EMPLOYEES		DEPARTMENTS		LOCATIONS	
LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID	LOCATION_ID	CITY
King	90	10	1700	1400	Southlake
Kochhar	90	20	1600	1600	South San Francisco
De Haan	90	50	1600	1700	Seattle
Hunold	60	60	1400	1800	Toronto
Ernst	60	60	2600	2500	Oxford
Lorentz	60	90	1700		
		10	1700		
		110	1700		
		190	1700		
Grant					
Whalen	10				
Hartstein	20				
Fay	20				
Higgins	110				
Gietz	110				

20 rows selected.

8 rows selected.

Uneori trebuie să efectueze (join) mai mult de două tabele. De exemplu, pentru a afișa numele, numele departamentului și orașul pentru fiecare angajat trebuie efectuat un join între tabelele EMPLOYEES, DEPARTMENTS și LOCATIONS.



```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id, l.city
FROM employees e, departments d, locations l
WHERE e.department_id=d.department_id and
d.location_id=l.location_id;
```

LAST_NAME	DEPARTMENT_NAME	CITY
Hunold	IT	Southlake
Ernst	IT	Southlake
Lorentz	IT	Southlake
Mourgos	Shipping	South San Francisco
Rais	Shippers	South San Francisco

Taylor	Sales	Oxford	
--------	-------	--------	--

Observație:



Întotdeauna numărul de condiții de join scrise în clauza **WHERE** trebuie să fie cu unu mai mic decât numărul de tabele scrise în clauza **FROM**. De exemplu, dacă numărul tabelelor este ,n', numărul condițiilor de join din clauza **WHERE** trebuie să fie ,n-1'.

8.4 Non-equi-join

EMPLOYEES		JOB_GRADES		
Last_Name	Salary	Grade	Lowest_sal	Highest_sal
King	24000	A	1000	2999
Kochhar	17000	B	3000	5999
De Haan	17000	C	6000	9999
Hunold	9000	D	10000	14999
Ernst	8000	E	15000	24999
Mourgos	5800	F	25000	40000

← salarul din tabelul EMPLOYEES este cuprins între salarul minim și maxim din tabelul JOB_GRADES

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Relația dintre tabelele EMPLOYEES și JOB_GRADE este de tip non-echijoin, adică nicio coloană din tabelul EMPLOYEES nu corespunde direct unei coloane din tabelul JOB_GRADE. Relația dintre cele două tabele este astfel: valorile din coloana SAL din EMPLOYEES sunt cuprinse între valorile din coloanele LOWEST_SAL și HIGHEST_SAL din tabelul JOB_GRADE. Relația (legătura) se obține folosind un operator, altul decât egal (=).

Extragerea înregistrarilor cu Non-echijoin-uri



```
SELECT e.last_name, e.salary, j.grade_level  
FROM employees e, job_grades j  
WHERE e.salary between j.lowest_sal and j.highest_sal;
```

Exemplul de sus creează un non-echijoin pentru a evalua gradul salarului unui angajat. Salarul trebuie să fie între orice pereche "cel mai mic"- "cel mai mare" a intervalor salariale.

LAST_NAME	SALARY	GRA
Matos	2500	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Kochhar	17000	E
De Haan	17000	E

20 rows selected.

Este important de remarcat că toți angajații apar doar odată atunci când această interogare este executată. Niciun angajat nu este repetat în listă și acest lucru are la bază două motive:

- Nicio linie din tabelul cu gradele salariale nu conține trepte salariale suprapuse (cu valorile pentru alte trepte salariale). Astfel, valoarea salariului unui angajat poate oscila numai între salariul minim și maxim din una din liniile tabelului ce conține treptele salariale.
- Toate salariile angajaților sunt încadrate între limitele date de treptele salariale. Astfel, nici un angajat nu câștiga mai puțin decât cea mai mică valoare din coloana LOWEST_SAL sau mai mult decât cea mai mare valoare conținută în coloana HIGHEST_SAL.



Observație: Se pot folosi și alți operatori, cum ar fi `<=` și `>=`, dar este mai simplu de folosit BETWEEN. Când se folosește BETWEEN trebuie specificată mai întâi valoarea minimă și apoi valoarea maximă. Aliasurile de tabele au fost specificate din motive de performanță și nu datorită unei posibile ambiguități.

8.5 Outer-join

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
90	Hunold
80	Ernst
10	Mavrodien
20	Harstein
20	Fay
110	Higgins
110	Gietz

Nici un angajat în departamentul 190

Returnarea înregistrărilor cu NoDirectMatch (Potrivire Indirectă) cu Joinuri externe.

Dacă o linie (înregistrare) nu satisfacă condiția de join, acea linie nu va apărea în rezultatul interogării. De exemplu, în condiția de echijoin a tabelelor EMPLOYEES și DEPARTMENTS, departamentul OPERATIONS nu va apărea pentru că nu lucrează nimeni în acel departament.



```
SELECT e.last_name, e.department_id, d.department_name  
FROM employees e, departments d  
WHERE e.department_id = d.department_id;
```

```
SELECT table1.column, table2.column  
FROM table1, table2  
WHERE table1.column(+) = table2.column;
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
...
Wiggins	110	Accounting
Gietz	110	Accounting



```
SELECT table1.column, table2.column  
FROM table1, table2  
WHERE table1.column(+) = table2.column;  
  
SELECT table1.column, table2.column  
FROM table1, table2  
WHERE table1.column = table2.column(+);
```

În sintaxă:

table1.column = este condiția de join a tabelelor;
table2.column(+) este simbolul pentru join extern; poate fi plasat în
oricare parte a condiției din clauza WHERE, dar nu în ambele
parti deodată. Plasați simbolul join extern după numele
coloanei din tabelul deficitar în informații.



Folosiți un join extern pentru a vedea liniile care nu îndeplinesc
condiția de join.

Operatorul join extern este semnul plus (+).

Linia (liniile) lipsă pot fi returnate dacă este folosit un join extern în
condiția de join. Operatorul este un semn "plus" scris între paranteze (+) care
este plasat lângă tabelul ce prezintă un deficit de informație. Acest operator are
efectul creării a unei sau mai multe lini împărtășite (nule), pentru fiecare linie din
tabelul non-deficient.



```
SELECT e.last_name, e.department_id, d.department_name  
FROM employees e, departments d  
WHERE e.department_id(+) = d.department_id;
```

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	60	Shipping
Rajs	60	Shipping
Pinggins	110	Contracting
Gietz	110	Accounting
		Contracting

20 rows selected.

Exemplul afișează numele angajatului, numărul și numele pentru toate departamentele. Departamentul Contracting, care nu are nici un angajat este de asemenea afișat.

Restrictii la Join extern



- Operatorul join extern poate să apară numai într-o singură parte a unei expresii și anume în partea deficitară în informații. El returnează acele linii din tabel care nu au corespondent direct în celalalt tabel.
- O condiție implicată într-un join extern nu poate folosi operatorul IN sau nu poate fi legată la o altă condiție prin operatorul OR.

8.6 Self – Join

Uneori este necesară alăturarea (join) unui tabel cu el însuși. De exemplu, pentru a găsi numele fiecărui angajat pentru un manager este necesar un self-join pe tabela EMPLOYEES. De exemplu, pentru a găsi numele managerului lui Whalen este nevoie să:

- o Îl găsiți pe Whalen în tabelul EMPLOYEES uitându-vă în coloana Last_name;
- o Găsiți numărul managerului pentru Whalen uitându-vă în coloana Manager_ID. Numărul managerului lui Whalen este 101.
- o Găsiți numele managerului cu EMPLOYEE_ID egal cu 101 uitându-vă în coloana LAST_NAME. Numărul angajatului Kochhar este 101 deci Kochhar este managerul lui Whalen.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

EMPLOYEES (WORKER)			EMPLOYEES (MANAGER)		
EMPLOYEE_ID	LAST_NAME	MANAGER_ID	EMPLOYEE_ID	LAST_NAME	
101	Kochhar	100	100	King	
102	De Haan	100	100	King	
124	Mourgos	100	100	King	
149	Zlotkey	100	100	King	
201	Hartstein	100	100	King	
200	Whalen	101	101	Kochhar	
206	Gietz		206	Higgins	

19 rows selected.

"MGR din tabelul WORKER este egal cu EMPNO din tabelul MANAGER"

În acest proces căutarea în tabel se execută de două ori. Prima dată se caută în tabel angajatul Whalen în coloana Last_name și găsește valoarea pentru Manager_ID care este egală cu 101. A doua oară se caută în coloana EMPLOYEE_ID pentru a găsi valoarea 101 pentru care în coloana Last_name se valoarea este Kochhar.

 **SELECT worker.last_name || ' works for '|| manager.last_name
FROM employees worker, employees manager
WHERE worker.manager_id = manager.employee_id;**

Exemplul de mai sus alătură tabela EMPLOYEES cu ea însăși. Pentru a simula cele două tabele în clauze FROM se folosesc două aliasuri, numite WORKER și MANAGER, pentru același tabel, EMPLOYEES.

În acest exemplu clauza WHERE conține join-ul care înseamnă "unde numărul managerului pentru un subaltern este identic cu numărul angajatului pentru acel manager".

W.LAST_NAME 'WORKSFOR' M.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Montgomery works for De Haan
Jay works for Hartstein
Gietz works for Higgins

19 rows selected.

8.7 Definirea join-urilor folosind sintaxa SQL 1999



```
SELECT table1.column, table2.column  
FROM table1  
[CROSS JOIN table2] |  
[NATURAL JOIN table2] |  
[JOIN table2 USING (column_name)] |  
[JOIN table2  
ON (table1.column_name = table2.column_name)] |  
[LEFT|RIGHT|FULL OUTER JOIN table2  
ON (table1.column_name = table2.column_name)];
```

În sintaxă:

table1.column - tabelul și coloana de unde se extrag datele;
CROSS JOIN - returnează produsul cartezian a celor două tabele;
NATURAL JOIN - face un join pe cele două tabele pe baza unei coloane comune;
JOIN table USING - column_name execută un echi-join bazat pe coloana specificată;
JOIN table ON table1.column_name = table2.column_name - execută un join bazat pe condiția specificată în clauza ON.

Crearea Cross Joins

Folosirea clauzei CROSS JOIN duce la apariția unui produs cartezian a datelor din două tabele.



```
SELECT last_name, department_name  
FROM employees  
CROSS JOIN departments;
```

Exemplul de mai sus duce la același rezultat dat de exemplul următor:

LAST_NAME	DEPARTMENT_NAME
Janstein	Marketing
Fay	Contracting
Higgins	Contracting
Gietz	Contracting

160 rows selected.



```
SELECT last_name, department_name FROM employees,  
departments;
```

Crearea Natural Joins

Clauza NATURAL JOIN are la bază toate coloanele din cele două tabele ce au același nume. La execuție se selectează rândurile din cele două tabele ce au valori egale în toatele coloanele comune. Dacă coloanele ce au același nume au tipuri diferite de date apare o eroare.

În versiunile Oracle premergătoare Oracle9i nu se putea face un join fără a specifica coloanele pe care să se execute join-ul. Începând cu această versiune este posibil ca join-ul să se execute complet automat folosind cuvintele cheie NATURAL JOIN care permit executarea join-ului pe baza coloanelor celor două tabele ce au același nume și același tip de dată.



```
SELECT department_id, department_name,  
location_id, city  
FROM departments NATURAL JOIN locations;
```

În exemplu tabela LOCATIONS este alăturată tabelei DEPARTMENT prin coloana comună LOCATION_ID care este singura coloană ce are același nume în ambele tabele. Dacă ar fi fost și alte coloane care să aibă același nume în ambele tabele ar fi fost folosite și acestea.

Equijoins



Natural join poate fi scris și ca un equijoin:

```
SELECT department_id, department_name,  
departments.location_id, city  
FROM departments, locations  
WHERE departments.location_id = locations.location_id;
```

Natural Joins cu clauza WHERE

Folosind clauza WHERE se pot impune restricții suplimentare în folosirea unui natural join. Exemplul de mai jos limitează rândurile afișate la cele ce corespund condiției ca department_ID să fie 20 sau 50.



```
SELECT department_id, department_name,  
location_id, city  
FROM departments  
NATURAL JOIN locations WHERE department_id IN (20, 50);
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Gietz	Contracting

160 rows selected.

Crearea de Joins folosind clauza USING

Dacă anumite coloane din tabele diferite au același nume dar nu au același tip de date, clauza NATURAL JOIN poate fi modificată prin folosirea clauzei USING pentru a specifica coloanele ce trebuie folosite în equijoin.

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
60	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1701	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

Notă:

- Când referiți coloana, oriunde în instrucțiunea SQL, nu trebuie să folosiți numele tabelei sau aliasul.
 - Clauzele NATURAL JOIN și USING sunt mutual exclusive

De exemplu următoarea instrucțiune este validă



```
SELECT l.city, d.department_name  
FROM locations l JOIN departments d USING (location_id)  
WHERE location_id = 1400;
```

Următoarea instrucțiune este invalidă deoarece coloana LOCATION_ID este calificată în clauza WHERE:

```
SELECT l.city, d.department_name  
  FROM locations l JOIN departments d USING (location_id)  
 WHERE d.location_id = 1400;  
ORA-25154: column part of USING clause cannot have qualifier
```

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

Aceeași restricție se aplica și la NATURAL JOIN.

Așadar coloanele ce au același nume în ambele tabele trebuie să fie scrise fără calificatori.



```
SELECT e.employee_id, e.last_name, d.location_id  
FROM employees e JOIN departments d  
USING (department_id);
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID
200	Whalen	1700
201	Hartstein	1800
202	Fay	1800
124	Mourgos	1500
141	Rajs	1500
142	Davies	1500
205	Higgins	1700
206	Gietz	1700

19 rows selected.

În exemplu se face un join pe coloana DEPARTMENT_ID între tabela EMPLOYEES și tabela DEPARTMENTS pentru a se afișa locația unde lucrează un angajat. Acest lucru poate fi scris și ca un equijoin:



```
SELECT employee_id, last_name,  
employees.department_id, location_id  
FROM employees, departments  
WHERE employees.department_id = departments.department_id;
```

Crearea de Joins folosind clauza ON

Condiția de join pentru un natural join este de fapt un equijoin al tuturor coloanelor ce au același nume. Pentru a specifica condiții arbitrate sau coloanele pe care se face join-ul se folosește clauza ON.

Clauza ON separă condiția de join de alte condiții de căutare și face codul mai ușor de înțeles.



```
SELECT e.employee_id, e.last_name, e.department_id,  
d.department_id, d.location_id  
FROM employees e JOIN departments d  
ON (e.department_id = d.department_id);
```

Clauza ON poate fi deasemeni folosită pentru a alătura coloane ce nu au același nume.



```
SELECT e.last_name emp, m.last_name mgr
FROM employees e JOIN employees m
ON (e.manager_id = m.employee_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	60	1500
142	Davies	50	50	1500
205	Higgins	110	110	1700
206	Gietz	110	110	1700

EMP	MGR
Kochhar	King
De Haan	King
Mourgos	King

Gietz	Higgins
-------	---------

19 rows selected.

Exemplul de mai sus este de fapt un self join pe tabela EMPLOYEES bazat pe legătura dintre coloanele EMPLOYEE_ID și MANAGER_ID.

Crearea de Three-Way Joins prin folosirea clauzei ON



```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l ON d.location_id = l.location_id;
```

Un join de tip three-way join este un join bazat pe trei tabele. În sintaxă conformă cu SQL: 1999 join-urile sunt executate de la stânga la dreapta, deci primul join executat este cel dintre EMPLOYEES și DEPARTMENTS. Prima condiție de join poate referenția coloanele din EMPLOYEES și DEPARTMENTS dar nu poate referenția coloane din LOCATIONS.

A doua condiție de join poate referenția coloane din toate cele trei tabele. Acest lucru poate fi scris ca un echijoin pe trei tabele.



```
SELECT employee_id, city, department_name
FROM employees, departments, locations
WHERE employees.department_id =
departments.department_id
AND departments.location_id = locations.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
100	Seattle	Executive
101	Seattle	Executive
102	Seattle	Executive
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
108	South San Francisco	Chimneys

206 |Seattle ... 19 rows selected.

INNER versus OUTER Joins

- În SQL: 1999, join-ul dintre două tabele ce returnează doar un singur rând comun este un inner join.
- Un join între două tabele ce returnează atât rezultatele unui inner join cât și rândurile ce nu se potrivesc cu tabela din stânga (sau dreapta) este un outer join la stânga (sau dreapta).
- Un join între două tabele ce returnează atât rezultatul unui inner join cât și rezultatul unui left join și right join este un full outer join.

Joins: Comparing SQL: 1999 to Oracle Syntax

Oracle	SQL: 1999
Equipjoin	Natural or Inner Join
Outerjoin	Left Outer Join
Selfjoin	Join ON
Nonequipjoin	Join USING
Cartesian Product	Cross Join



LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
Gietz	110	Accounting
Ernst	80	IT
Grant		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Morgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
Gietz	110	Accounting
		Contracting

20 rows selected.

Acest exemplu afișează toate rândurile din tabela EMPLOYEES, care este tabela din stânga, chiar dacă nu este nici o „potrivire” în tabela DEPARTMENTS. Această interogare se poate scrie și astfel :



```

SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id (+) = e.department_id;
RIGHT OUTER JOIN
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id);

```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Morgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
Gietz	110	Accounting
		Contracting

20 rows selected.

Acest exemplu afișează toate rândurile din tabela DEPARTMENTS, care este tabela din dreapta, chiar dacă nu este nici o „potrivire” cu tabela EMPLOYEES. Această interogare se poate scrie și astfel :



```

SELECT e.last_name, e.department_id,d.department_name
FROM employees e, departments d
WHERE d.department_id = e.department_id (+);

```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

FULL OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
FULL OUTER JOIN departments d ON (e.department_id = d.department_id);
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Abel	80	Sales
Davies	50	Shipping
De Haan	90	Executive
Ernst	60	IT
Fay	20	Marketing
Gietz	110	Accounting
Grant		
Hartstein	20	Marketing

Zlotkey	80	Sales
		Contracting

21 rows selected.

Această interogare afișează toate rândurile din tabela EMPLOYEES chiar dacă nu există nici o corespondență cu datele din tabela DEPARTMENTS. Deasemeni sunt afișate toate rândurile din tabela DEPARTMENTS chiar dacă nu există nici o corelație cu datele din tabela EMPLOYEES.



Condiții suplimentare

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM employees e JOIN departments d
ON (e.department_id = d.department_id) AND e.manager_id = 149;
```

Se pot adăugă în clauza WHERE condiții suplimentare aşa cum se arată în exemplul anterior. Datele rezultate în urma join-ului dintre tabelele EMPLOYEES și DEPARTMENTS sunt restricționate de condiția impusă și anume valoarea coloanei manager_ID din tabela employees sa fie egală cu 149.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel	80	80	2500
176	Taylor	80	80	2500

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



8.8 Exerciții

1. Scrieți o interogare care să afișeze numele, numărul departamentului și numele departamentului pentru toți angajații.

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
Vargas	50	Shipping
Leverett	60	IT
Wing	110	Accounting
Gietz	110	Accounting

2. Scrieți o interogare care să afișeze meserile distințe (jobs) și numele departamentului pentru toți angajații din departamentul 30. Includeți și locația pentru departamentul 90.

JOB_ID	LOCATION_ID
SA_MAN	2500
SA_REP	2500

3. Scrieți o interogare care afișează numele angajatului, numele departamentului, locația și orașul tuturor angajaților care câștiga un comision.

LAST_NAME	DEPARTMENT_NAME	LOCATION_ID	CITY
Zlotkey	Sales	2500	Oxford
Abel	Sales	2500	Oxford
Taylor	Sales	2500	Oxford

4. Afișați numele angajatului și numele departamentului pentru toți angajații care au un „a” în numele lor. Salvați tabelul SQL într-un fișier numit p4q4.sql.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	DEPARTMENT_NAME
Whalen	Administration
Hartstein	Marketing
Fay	Marketing
Rajs	Shipping
Davies	Shipping
Matos	Shipping
Vargas	Shipping
Taylor	Sales
Kochhar	Executive
De Haan	Executive

10 rows selected.

4. Scrieți o interogare care afișează numele, meseria, numărul departamentului și numele departamentului pentru toți angajații care lucrează în Toronto.

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
Hartstein	MK_MAN	20	Marketing
Fay	MK_REP	20	Marketing

5. Afișați numele și marca angajatului împreună cu numele și marca managerului acestuia. Etichetați coloanele Employee, Emp#, Manager, Mgr#. Salvați interogarea SQL într-un fișier numit p4q6.sql.

Employee	EMP#	Manager	Mgr#
Kochhar	101	King	100
De Haan	102	King	100
Mourgos	124	King	100
Zlotkey	149	King	100

Abel	174	Zlotkey	149
Taylor	176	Zlotkey	149
Grant	178	Zlotkey	149
Fay	202	Hartstein	201
Gietz	206	Higgins	205

19 rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

6. Modificați p4q6.sql pentru a afișa toți angajații incluzând pe King care nu are manager.

Employee	EMP#	Manager	Mgr#
King	100		
Kochhar	101	King	100
De Haan	102	King	100
Hunold	103	De Haan	102
Ernst	104	Hunold	103
Lorentz	107	Hunold	103
Mourgos	124	King	100
...

Higgins	205	Kochhar	101
Gietz	206	Higgins	205

20 rows selected.

7. Creați o interogare care va afișa pentru toți angajații numele angajatului, numărul departamentului și toți angajații care lucrează în același departament (colegii lui). Denumiți ultima coloană Coleg. Aveți grija ca în interogare să nu apară un angajat coleg cu el însuși.

DEPARTMENT	EMPLOYEE	COLLEAGUE
20	Fay	Hartstein
20	Hartstein	Fay
50	Davies	Matos
50	Davies	Mourgos
50	Davies	Rajs
50	Davies	Vargas
50	Matos	Davies
50	Matos	Mourgos
50	Matos	Rajs
50	Matos	Vargas

110	Gietz	Higgins
110	Higgins	Gietz

42 rows selected.

8. Afișați structura tabelului JOB_GRADEs. Creați o interogare care va afișa numele, meseria, numele departamentului, salariul și treapta de salarizare pentru toți angajații.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Name	Null?	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRA
Matos	ST_CLERK	Shipping	2600	A
Vargas	ST_CLERK	Shipping	2500	A
Lorentz	IT_PROG	IT	4200	B
Mourgos	ST_MAN	Shipping	5800	B
Rajs	ST_CLERK	Shipping	3500	B
Davies	ST_CLERK	Shipping	3100	B
Whalen	AD_ASST	Administration	4400	B

De Haan	AD_VP	Executive	17000	E
---------	-------	-----------	-------	---

19 rows selected.

9. Creați o interogare care afișează numele și data angajării pentru lucrătorii angajați după data de angajare a lui Davies.

LAST_NAME	HIRE_DATE
Lorentz	07-FEB-99
Mourgos	16-NOV-99
Matos	15-MAR-98
Vargas	09-JUL-98
Zlotkey	29-JAN-00
Taylor	24-MAR-98
Grant	24-MAY-99
Fay	17-AUG-97

8 rows selected.

10. Afișați toate numele angajaților și data angajării împreună cu numele managerilor și data lor de angajare, pentru toți cei care au fost angajați înaintea managerilor lor. Etichetați coloanele Employee, respectiv Emp, Hiredate, Manager, și Mgr Hiredate.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Employee	Emp Hired	Manager	Mgr Hired
Whalen	17-SEP-87	Kochhar	21-SEP-89
Hunold	03-JAN-90	De Haan	13-JAN-93
Rajs	17-OCT-95	Mourgos	16-NOV-99
Davies	29-JAN-97	Mourgos	16-NOV-99
Matos	15-MAR-98	Mourgos	16-NOV-99
Vargas	09-JUL-98	Mourgos	16-NOV-99
Abel	11-MAY-96	Zlotkey	29-JAN-00
Taylor	24-MAR-98	Zlotkey	29-JAN-00
Grant	24-MAY-99	Zlotkey	29-JAN-00

9 rows selected.

11. Creați o interogare care afișează numele angajaților și salariile indicate prin asteriscuri. Fiecare asterisc înseamnă 100 \$. Sortați datele în ordinea descendentă a salariilor. Etichetați coloana EMPLOYEE_AND_THEIR_SALARIES.

EMPLOYEE_AND_THEIR_SALARIES	
KING	*****
FORD	*****
SCOTT	*****
JONES	*****
BLAKE	*****
CLARK	*****
ALLEN	*****
TURNER	*****
MILLER	*****
MARTIN	*****
WARD	*****
ADAMS	*****
JAMES	*****
SMITH	*****

14 rows selected.

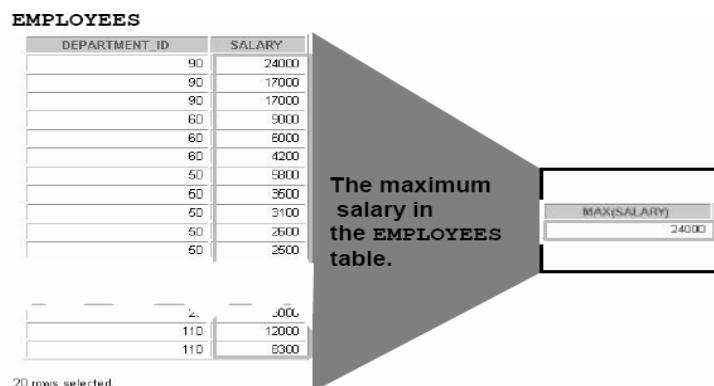
Capitolul 9 Folosirea funcțiilor de grup

Obiective:

- Identificarea funcțiilor de grup disponibile;
- Descrierea folosirii funcțiilor de grup;
- Gruparea datelor folosind clauza **Group By**;
- Includerea sau excluderea liniilor grupate folosind clauza **HAVING** .

9.1 Ce sunt funcțiile de GRUP ?

Funcțiile de grup operează pe seturi de linii oferind un singur rezultat pentru tot grupul.



Spre deosebire de funcțiile de un singur rând, funcțiile de grup operează pe seturi de rânduri pentru a da un singur rezultat unui grup. Aceste seturi pot fi întregul tabel sau tabelul împărțit la rândul lui în grupuri. Funcțiile de grup sunt următoarele :

- **AVG**
- **COUNT**
- **MAX**
- **MIN**
- **STDDEV**
- **SUM**
- **VARIANCE**

Fiecare din funcții acceptă/primește un argument. Următorul tabel identifică opțiunile pe care le puteți folosi în sintaxă.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Funcție	Descriere
AVG([DISTINCT ALL]n)	Valoarea medie a lui "n", ignorând valorile nule;
COUNT({* [DISTINCT ALL]expr})	Numărul de rânduri, unde expresia evaluează altceva decât valori nule. Numără toate rândurile selectate folosind *, inclusiv duplicatele și rândurile cu valori nule;
MAX([DISTINCT ALL]expr)	Valoarea maximală a expresiei, ignorând valorile nule;
MIN([DISTINCT ALL]expr)	Valoarea minimă a expresiei, ignorând valorile nule;
STDDEV([DISTINCT ALL]x)	Abaterea standard a lui "n", ignorând valorile nule;
SUM([DISTINCT ALL]n)	Suma valorilor lui "n", ignorând valorile nule;
VARIANCE([DISTINCT ALL]x)	Variația lui "n", ignorând valorile nule;



```

SELECT      coloana,  funcție_de_grup(coloana)
FROM        tabela
[WHERE      condiție]
[ORDER BY   coloana];
  
```

Sfaturi pentru folosirea funcțiilor de grup:



- **DISTINCT** face ca funcția să ia în considerare numai valorile distincte. **ALL** ia în considerație fiecare valoare, inclusiv valorile duble. "**ALL**" este implicit și deci nu mai trebuie specificat.
- Tipurile de date pentru argumente pot fi : **CHAR**, **VARCHAR2**, **NUMBER** sau **DATE**.
- Toate funcțiile de grup, cu excepția **COUNT(*)** ignora valorile nule. Pentru a înlocui o valoare cu valori nule folosiți funcția **NVL**, **NVL2** sau **COALESCE**.

9.2 Folosirea funcțiilor AVG, SUM, MIN, MAX

Funcțiile **AVG** sau **SUM** se pot folosi pentru date de tip numeric.



```
SELECT AVG(salary), MAX(salary),  
MIN(salary), SUM(salary)  
FROM employees  
WHERE job_id LIKE "%REP%";
```

Avg(Salary)	Max(Salary)	Min(Salary)	Sum(Salary)
6150	11000	6000	32600

Funcțiile **AVG**, **SUM**, **MIN** și **MAX** se pot folosi pentru coloanele care pot stoca date numerice. Exemplul de mai sus afișează media, maximul, minimul și suma salariilor lunare pentru toți vânzătorii.

Funcțiile **MIN** și **MAX** se pot folosi pentru orice tip de date.



```
SELECT MIN(hire_date), MAX(hire_date)  
FROM employees;
```

MIN(HIRE_DATE)	MAX(HIRE_DATE)
17-JUN-87	9-JAN-00

Exemplul următor afișează numele pentru primul și ultimul angajat din lista alfabetică a tuturor angajaților.



```
SELECT MIN(last_name), MAX(last_name)  
FROM employees;
```

MIN(LAST_NAME)	MAX(LAST_NAME)
Abel	Zlotkey

9.3 Folosirea funcției COUNT

COUNT(*) returnează numărul de linii dintr-o tabelă.



```
SELECT COUNT(*)  
FROM employees  
WHERE department_id = 50;
```

Funcția **COUNT** are trei formate :

- **COUNT(*)** - întoarce numărul de rânduri în tabel, incluzând rândurile duble și rândurile conținând valori nule;
- **COUNT(expr)** - întoarce numărul rândurilor nenule din coloana identificată prin expr.;
- **COUNT(DISTINCT expr)** - întoarce numărul rândurilor unice nenule din coloana identificată prin expr.

Exemplul de mai jos afișează numărul angajaților din departamentul 80 care pot să câștige un comision.



```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

COUNT(COMMISSION_PCT)

3

Exemplu: Afișați numărul departamentelor din tabelul EMPLOYEES.

```
SELECT COUNT(department_id)
FROM employees;
```

COUNT(DEPARTMENT_ID)

19

Afișați numărul departamentelor distincte din tabelul EMPLOYEES.

```
SELECT COUNT(DISTINCT department_id)
FROM employees;
```

COUNT(DISTINCTDEPARTMENT_ID)

7

9.4 Funcțiile de grup și valorile Null

Funcțiile de grup ignoră valorile null din coloană.



```
SELECT AVG(commission_pct)
FROM employees;
```

Facultatea de Automatică și Calculatoare Iași **Baze de date – lucrări practice**

Avg(COMMISSION_PCT)
.2125

Toate funcțiile grup, cu excepția COUNT(*), ignoră valorile nule din coloană. În exemplul de mai sus media este calculată **doar** pe baza rândurilor din tabel în care coloana COMMISSION_PCT conține o valoare validă. Media este calculată ca sumă a comisioanelor plătite către toți angajații, împărțită la numărul angajaților care primesc comision.

Funcția NVL forțează funcțiile de grup să includă valori nule.

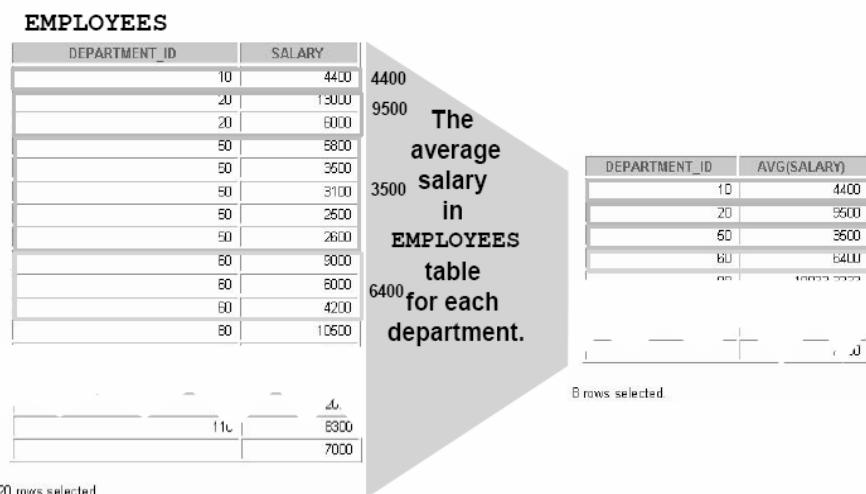


```
SELECT AVG(NVL(commission_pct, 0))  
FROM employees;
```

AVG(NVL(COMMISSION_PCT,0))
.0425

Funcția **NVL** forțează funcțiile grup să includă valori nule. În exemplul de mai sus media este calculată pe baza tuturor rândurilor din tabel indiferent dacă în coloana COMMISSION_PCT sunt stocate valori nule. Media este calculată ca un comision total plătit tuturor angajaților, împărțit la numărul total al angajaților companiei.

9.5 Crearea grupurilor de date



Până acum toate funcțiile grup au tratat tabelul ca fiind un larg grup de informații. Uneori însă tabelul trebuie împărțit în grupuri mai mici de informații.

Aceasta se poate face folosind clauza **GROUP BY**.



```
SELECT column, group_function(column)
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

În sintaxă :**group by expresion** specifică coloanele ale căror valori determină bazele pentru gruparea rândurilor.

Clauza **GROUP BY** se poate folosi pentru a împărți rândurile din tabel în grupuri, lucru ce permite folosirea funcțiilor de grup pentru a întoarce sumarul informației pentru fiecare grup.



Sfaturi

- Dacă se include o funcție de grup într-o clauza **SELECT** nu se pot selecta rezultatele individuale **decât** dacă coloana individuală apare în clauza **GROUP BY**. Dacă coloana respectivă nu este inclusă în clauza **Group By** va fi generat un mesaj de eroare ;
- Clauza **WHERE** exclude rândurile înainte de a formarea grupurilor.
- Implicit, rândurile sunt sortate în ordinea ascendentă a coloanelor incluse în lista **GROUP BY**. Se poate specifica o altă ordine folosind clauza **ORDER BY**.



Toate coloanele din lista **SELECT** care nu sunt funcții de grup trebuie să fie menționate în clauza **GROUP BY**.



```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
30	3500
40	6400
50	10033.3333
60	19333.3333
70	10150
80	7000
90	
110	

Exemplul de mai sus afișează numărul departamentului și media salariilor pentru fiecare departament. Iată cum este evaluată declarația **SELECT** de mai sus, conținând o clauză **GROUP BY**:

- Clauza **SELECT** specifică coloanele care să fie afișate, adică numărul departamentului și media tuturor salariilor din grupul specificat în clauza **GROUP BY**.
- Clauza **FROM** specifică tabelul pe care baza de date trebuie să-l acceseze: tabelul EMPLOYEES.
- Clauza **WHERE** specifică liniile ce trebuie incluse. Dacă nu există nici o clauză **WHERE**, implicit toate rândurile sunt incluse.
- Clauza **GROUP BY** specifică modul de grupare pentru rânduri. Rândurile sunt grupate după numărul departamentului, deci funcția **AVG** care este aplicată coloanei salariilor va calcula *media salariilor pentru fiecare departament*.

Coloanele **GROUP BY** care nu sunt în lista **SELECT**:



```
SELECT AVG(salary)
FROM employees
GROUP BY department_id;
```

AVG(SALARY)
4400
9500
3500
6400
10033.3333
19333.3333
10150
7000

8 rows selected.

Coloana specificată în clauza **GROUP BY** nu trebuie să fie obligatoriu menționată în clauza **SELECT**. De exemplu, declarația **SELECT** de mai sus afișează media salariilor pentru fiecare departament fără să afișeze numărul departamentului respectiv. Totuși, fără numărul departamentului, rezultatele nu afișează datele în mod semnificativ.

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

Se poate folosi funcția de grupare, în exemplu avg(sal), în clauza **ORDER BY**.



```
SELECT department_id, AVG(salary)  
FROM employees  
GROUP BY department_id  
ORDER BY AVG(salary);
```

DEPARTMENT_ID	AVG(SALARY)
50	3500
10	4400
60	6400

8 rows selected.

9.6 Gruparea datelor după mai multe coloane

EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	3500
50	ST_CLERK	3100
50	ST_CLERK	2600
50	ST_CLERK	2500
50	ST_MAN	5800
60	IT_PROG	9000
60	IT_PROG	6000
60	IT_PROG	4200
80	SA_MAN	10500
80	SA_REP	11000

20 rows selected.

Add up the salaries in the EMPLOYEES table for each job, grouped by department.

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

Grupuri în grupuri

Câteodată sunt necesare rezultate pentru grupuri formate din subgrupuri. Exemplul de mai sus arată un raport care afișează totalul salariilor ce au fost plătite pentru fiecare nume de funcție, din fiecare departament.

Tabelul EMPLOYEES este grupat mai întâi după numărul departamentului și apoi această grupare se detaliază după numele funcției. De exemplu, doi funcționari din departamentul 20 sunt grupați împreună și se afișează un singur rezultat (salariul total).

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



```
SELECT department_id dept_id, job_id, SUM(salary)
FROM employees
GROUP BY department_id, job_id;
```

DEPT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	24000

[SA REP]

13 rows selected

Pentru o obține rezultatele finale pentru grupuri și subgrupuri trebuie să indicate mai multe coloane în clauza **GROUP BY**. Ordinea implicită de ordonare a rezultatelor poate fi determinată prin ordinea de scriere a coloanelor din clauza **GROUP BY**. Iată cum se evaluatează declarația SELECT de mai sus, care conține o clauza **GROUP BY**:

- Clauza **SELECT** specifică coloanele ce trebuie afișate:
 - Numărul departamentului din tabelul EMPLOYEES;
 - Numele funcției din tabelul EMPLOYEES;
 - Suma tuturor salariilor din grupul specificat în clauza GROUP BY.
- Clauza **FROM** specifică tabelul pe care baza de date trebuie să-l acceseze : tabelul EMPLOYEES;
- Clauza **GROUP BY** specifică cum trebuie grupate rândurile :
 - Rândurile se grupează mai întâi după numărul departamentului.
 - Apoi, din grupurile formate după numărul departamentului, se grupează rândurile după numele funcției. Deci funcția SUM este aplicată coloanei salariilor pentru toate numele de funcții din fiecare grup format după numărul departamentului.

9.7 Interogări ilegale în folosirea funcțiilor de grup



Orice coloană sau expresie din lista **SELECT** care nu este o funcție de grup trebuie să fie specificată în clauza **GROUP BY**.



```
SELECT department_id, COUNT(last_name)  
FROM employees;
```

Exemplul de mai sus generează eroarea:

```
SELECT department_id, COUNT(last_name)
```

*

ERROR at line 1:

ORA-00937: not a single-group group function

Column missing

Ori de cate ori se folosește o combinație de coloane individuale (*department_id*) și funcții grup (*COUNT(last_name)*) în aceeași declarație **SELECT**, trebuie inclusă o declarație (**GROUP BY**) care să specifice coloanele individuale (în acest caz *department_id*).

Dacă clauza **GROUP BY** lipsește, atunci apare mesajul de eroare "not a single-group function" și un asterisc (*) care indică coloana greșită. Eroarea de mai sus se repară adăugând clauza **(GROUP BY)**.

```
SELECT department_id, count(last_name)  
FROM employees  
GROUP BY department_id;
```

DEPARTMENT_ID	COUNT(LAST_NAME)
10	1
20	2

8 rows selected.



Orice coloană sau expresie din lista **SELECT** care nu este o funcție de grup trebuie să fie specificată în clauza **GROUP BY**.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



- Nu puteți folosi clauza **WHERE** pentru restricționarea grupurilor.
- Folosiți clauza **HAVING** pentru restricționarea grupurilor.
- Nu puteți folosi funcții de grup în clauza **WHERE**.



```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 8000
GROUP BY department_id;
WHERE AVG(salary) > 8000
*
```

*ERROR at line 3:
ORA-00934: group function is not allowed here*

Clauza **WHERE** nu poate fi folosită pentru a restricționa datele ce intra în componența grupurilor. Declarația **SELECT** de mai sus generează o eroare deoarece se folosește clauza **WHERE** pentru a restricționa afișarea mediei salariilor din acele departamente care au un salarior mediu mai mare de 8000 \$.

Eroarea de mai sus poate fi corectată prin folosirea clauzei **HAVING** pentru restricționarea datelor după formarea grupurilor.



```
SELECT department_id, AVG(salary)
FROM employees
HAVING AVG(salary) > 8000
GROUP BY department_id;
```

DEPARTMENT_ID	AVG(SALARY)
20	9500
80	10033.3333
90	19333.3333
110	10150

9.8 Excluderea rezultatelor obținute folosind clauza Group

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
60	10500
80	8800

The maximum salary per department when it is greater than \$10,000.

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

20 rows selected.

În același mod în care se folosește clauza **WHERE** pentru restricționarea rândurile selectate se poate folosi clauza **HAVING** pentru restricționarea datelor rezultate din grupuri. Pentru a afla salariul maxim pentru fiecare departament, doar pentru departamentele care au salariul maxim mai mare de 10.000\$, sunt necesare următoarele:

1. Găsirea salariul mediu pentru fiecare departament grupând după numărul departamentului.
2. Restricționarea grupurilor la acele departamente ce au salariu maxim mai mare de 10.000 \$.

Excluderea rezultatelor date de Group BY : Clauza HAVING

Folosirea clauzei **HAVING** pentru restricționarea datelor grupurilor presupune că :

- Rândurile să fie grupate.
- Clauza **GROUP BY** să fie aplicată.
- Grupurile care îndeplinesc condiția din clauza **HAVING** să fie afișate.

**SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]**

[ORDER BY column];

În sintaxă, *group condition* restricționează grupurile de rânduri raportate la acele grupuri a căror condiție specificată este TRUE (adevarat).

Facultatea de Automatică și Calculatoare Iași **Baze de date – lucrări practice**

Clauza **HAVING** poate fi folosită pentru a specifica care grupuri trebuie afișate. De aceea, mai întâi se restricționează grupurile pe baza informațiilor totale și apoi se exclud cele ce nu corespund condițiilor din clauza HAVING.



Serverul Oracle desfășoară următorii pași la folosirea clauzei **HAVING**:

- Se grupează rândurile;
- Funcția de grup se aplică grupului;
- Se afișează grupurile care îndeplinesc criteriul din clauza **HAVING**.



Clauza **HAVING** poate precede clauza **GROUP BY**, dar este recomandat scrierea mai întâi a clauzei **GROUP BY** deoarece este mai logic. Grupurile sunt formate și funcțiile grup sunt calculate înainte de aplicarea clauzei **HAVING** pentru grupurile din lista **SELECT**.



```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary)>10000;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

Exemplul de mai sus afișează numărul departamentului și salariul maxim pentru acele departamente la care salariul minim este mai mare de 10.000 \$.



Se poate folosi clauza **GROUP BY** fără a se folosi o funcție de grup în lista **SELECT**. Dacă se restricționează rândurile în baza unei funcții de grup, este obligatoriu necesară o clauza **GROUP BY** și o clauza **HAVING**.



Exemplul de mai jos afișează numerele departamentelor și salariul mediu la acele departamente la care salariul minim este mai mare de 10.000 \$.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING min(salary)>10000;
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

DEPARTMENT_ID	AVG(SALARY)
20	9500
80	10033.3333
90	19333.3333
110	10150



Exemplul de mai jos afișează numele funcției și totalul salariului lunar pentru fiecare nume de funcție, cu un total al statului de plată depășind 13.000 \$. Exemplul exclude vânzătorii și sortează lista după salariul lunar total.

```
SELECT job_id, SUM(salary) PAYROLL  
FROM employees  
WHERE job_id NOT LIKE "%REP%"  
GROUP BY job_id  
HAVING SUM(salary) > 13000  
ORDER BY SUM(salary);
```

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

9.9 Imbricarea funcțiilor de grup



Afișarea valorii maxime a salariului mediu.

```
SELECT MAX(AVG(salary))  
FROM employees GROUP BY department_id;
```

MAX(AVG(SALARY))
19333.3333

Funcțiile de grup pot fi imbricate pe oricâte nivele de adâncime. Exemplul de mai sus afișează salariul mediu maxim.

9.10 GROUP BY cu operatorii ROLLUP și CUBE

Obiective:

- Folosirea operatorului **ROLLUP** pentru a obține subtotaluri;
- Folosirea operatorului **CUBE** pentru a obține valori de intersecție;
- Folosirea funcției **GROUPING** pentru identificarea valorilor rândurilor create de **ROLLUP** sau **CUBE**;
- Folosirea **GROUPING SETS** pentru a obține un singur set de date.

Facultatea de Automatică și Calculatoare Iași **Baze de date – lucrări practice**

Operatorii **ROLLUP** și **CUBE** trebuie specificați în cadrul clauzei **GROUP BY**. Gruparea cu **ROLLUP** duce la obținerea unei set de rezultate ce conține pe lângă valorile pentru rândurile grupate în mod obișnuit și rândurile pentru subtotal.

Gruparea cu folosirea operatorului **CUBE** duce la gruparea rândurilor selectate pe baza valorilor tuturor combinațiilor posibile ale expresiilor specificate și returnează un singur rând cu informații totale pentru fiecare grup. Se poate folosi operatorul **CUBE** pentru a obține rânduri de tip cross-tabulation.



Observații:

La folosirea operatorilor **ROLLUP** și **CUBE** trebuie să vă asigurați că acele coloane ce sunt după clauza **GROUP BY** au sens, ca există o relaționare reală între ele. În caz contrar operatorii vor da rezultate irelevante.

Operatorii **ROLLUP** și **CUBE** sunt disponibili începând cu versiunea Oracle8i.

9.11 Operatorul ROLLUP



```
SELECT [column,] group_function(column)...
FROM table
[WHERE condition]
[GROUP BY [ROLLUP] group_by_expression]
[HAVING having_expression];
[ORDER BY column];
```

ROLLUP este o extensie a clauzei **GROUP BY**. Folosirea operatorului **ROLLUP** duce la obținerea unor rezultate cumulative cum ar fi subtotalurile.

ROLLUP poate fi folosit în scrierea rapoartelor, graficelor, etc. pentru a obține elemente de statistică și totaluri din seturile de rezultate.

Operatorul **ROLLUP** creează grupuri prin mișcarea într-o direcție, de la stânga la dreapta, de-a lungul listei de coloane specificate în clauza **GROUP BY** după care aplică funcția de agregare acestor grupări.



Observații:

Pentru a produce subtotaluri cu n dimensiuni, unde n este numărul de coloane specificat în clauza **GROUP BY** fără a folosi operatorul **ROLLUP**, trebuie unite, folosind operatorul **UNION ALL** n+1 fraze **SELECT**. Acest lucru face ca execuția interogării sa fie ineficientă deoarece fiecare frază **SELECT** necesită un acces la tabel.

Operatorul **ROLLUP** obține aceste rezultate cu un singur acces la tabel, motiv pentru care folosirea operatorului este foarte eficientă atunci când sunt

implicate multe coloane în producerea subtotalurilor.



EXEMPLU

```
SELECT department_id, job_id,sum(salary)
FROM employees WHERE department_id < 60
GROUP BY ROLLUP(department_id, job_id)
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
	10		4400
	20	MK_MAN	13000
	20	MK_REP	6000
	20		19000
	60	ST_CLERK	11700
	50	ST_MAN	5800
	50		17500
			40900

9 rows selected.

În exemplul de mai sus se afișează :

- Salariul total pentru fiecare JOB_ID din fiecare departament pentru acele departamente a căror ID este mai mic de 60, în conformitate cu gruparea pe departamente specificată în clauza **GROUP BY** (eticheta 1)
- Operatorul **ROLLUP** afișează:
 - Salariul total pentru acele departamente care au ID-ul mai mic de 60 (eticheta 2);
 - Salariul total pentru toate departamentele care au ID-ul mai mic de 60, în funcție de job_ID (eticheta 3);

Toate rândurile etichetate cu 1 sunt rânduri „normale” iar cele indicate cu 2 și 3 sunt rânduri „supratotalizatoare”.

Operatorul **ROLLUP** creează subtotaluri care se desfășoară începând de la nivelul cel mai detaliat până la un total general, în funcție de criteriile de grupare specificate în clauza **GROUP BY**. Se calculează mai întâi valorile totale standard pentru grupul specificat în clauza **GROUP BY** (în exemplu, suma salariilor pe fiecare funcție în cadrul departamentului), apoi se creează în mod progresiv subtotalul de nivel înalt, mutându-se de la dreapta la stânga în cadrul listei de valori de la clauza **GROUP BY** (în exemplu, este calculată mai întâi suma salariilor pe fiecare departament urmată de suma salariilor pentru toate departamentele).

Dacă pentru operatorul **ROLLUP** din clauza **GROUP BY** sunt specificate

n coloane, rezultatul operației va avea $n + 1$ grupuri (în exemplu $2 + 1 = 3$ grupuri).

Rândurile bazate pe prima expresie din cele n specificate se numesc rânduri „obișnuite” iar celelalte se numesc rânduri „supratotalizatoare”.

9.12 Operatorul CUBE



```
SELECT [column,] group_function(column)...
FROM table
[WHERE condition]
[GROUP BY [CUBE] group_by_expression]
[HAVING having_expression];
[ORDER BY column];
```

Operatorul **CUBE** este un „comutator” suplimentar al clauzei **GROUP BY** dintr-o fraza **SELECT**. Operatorul **CUBE** poate fi folosit împreună cu toate funcțiile de grup inclusiv **AVG**, **SUM**, **MAX**, **MIN** și **COUNT**. El este folosit pentru a obține seturi de rezultate care sunt în mod obișnuit folosite în rapoartele de tip „cross-tab”. Pe când **ROLLUP** duce la obținerea doar a unei fracții din numărul total de combinații posibile pentru subtotaluri, **CUBE** produce subtotaluri pentru toate combinațiile posibile ale grupării specificate în clauza **GROUP BY** precum și un total general.

Operatorul **CUBE** este folosit împreună cu o funcție de grup pentru a generaliza rânduri suplimentare într-un set de rezultate. Coloanele incluse în clauza **GROUP BY** sunt referite încrușiat în vederea generării unui superset de grupuri. Funcția de grup specificată în listă este aplicată acelor grupuri pentru a produce valori de total pentru rânduri suplimentare „supratotalizatoare”. Numărul grupurilor suplimentare din setul de rezultate este dat de numărul de coloane incluse în clauza **GROUP BY**.

De fapt, orice combinație posibilă a coloanelor sau expresiilor menționate în clauza **GROUP BY** este folosită pentru a produce supratotaluri. Dacă în clauza **GROUP BY** sunt n coloane sau expresii, vor fi 2^n combinații de supratotaluri posibile. Matematic, aceste combinații formează un cub n -dimensional, motiv pentru care operatorul este numit astfel.

Prin folosirea aplicațiilor sau elementelor de programare, aceste valori supratotalizatoare pot fi încărcate în grafice care convertează datele în elemente vizuale.

 Exemplu:

```
SELECT department_id, job_id, SUM(salary)
FROM employees WHERE department_id < 60
GROUP BY CUBE (department_id, job_id);
```

În exemplul de mai sus rezultatul interogării se interpretează astfel:

- Salariul total pentru fiecare funcție din cadrul departamentului (pentru acele departamente ce au ID-ul mai mic de 50) este afișat de clauza **GROUP BY** (eticheta 1);
- Salariul total pentru acele departamente ce au ID-ul mai mic de 50 (eticheta 2);
- Salariul total pentru fiecare funcție indiferent de departament (eticheta 3);
- Salariul total pentru acele departamente ce au ID-ul mai mic de 50, indiferent

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
10		4400
20	MK_MAN	13000
20	MK_REP	6000
20		19000
50	ST_CLERK	11700
50	ST_MAN	5800
50		17500
50	AD_ASST	4400
	MK_MAN	13000
	MK_REP	6000
	ST_CLERK	11700
	ST_MAN	5800
		46300

14 rows selected.

de funcție (eticheta 4).

În exemplu, toate rândurile etichetate cu 1 sunt rânduri obișnuite, toate rândurile etichetate cu 2 și 4 sunt rânduri totalizatoare iar rândurile etichetate cu 3 sunt rânduri ce conțin valori „încrucișate”.

Operatorul **CUBE** a efectuat și operația pe care un operator de tip **ROLLUP** ar fi executat-o pentru a afișa subtotalurile și salariul total pentru acele departamente a căror ID este mai mic, indiferent de valoarea din coloana **JOB_ID**. Suplimentar operatorul **CUBE** afișează salariul total pentru fiecare funcție, indiferent de departament.



Notă:

Similar cu operatorul **ROLLUP**, ce generează subtotaluri pe n dimensiuni (unde n este numărul de coloane din clauza **GROUP BY**),

dacă nu s-ar folosi operatorul **CUBE** ar fi fost necesare 2n fraze **SELECT** care să fie legate cu **UNION ALL**. Astfel, o ieșire cu 3 dimensiuni ar necesita $2^3 = 8$ fraze **SELECT** care să fie unite cu **UNION ALL**.

9.13 Funcția GROUPING

Funcția **GROUPING** poate fi folosită împreună cu operatorul **CUBE** sau **ROLLUP**. Prin folosirea ei se pot crea grupuri care să formeze subtotaluri într-un rând și se pot deosebi valorile de **NULL** stocate de valorile de **NULL** create de **ROLLUP** sau **CUBE**. Funcția întoarce valoarea 0 sau 1.



```
SELECT [column,] group_function(column).., GROUPING(expr)
FROM table [WHERE condition]
[GROUP BY [ROLLUP][CUBE] group_by_expression]
[HAVING having_expression];
[ORDER BY column];
```

Funcția **GROUPING** poate fi folosită împreună cu operatorul **CUBE** sau **ROLLUP** pentru a ajuta la înțelegerea modalității de obținere a valorilor de total.

Funcția **GROUPING** folosește o singură coloană drept argument. Valoarea pentru **expr** din sintaxa trebuie să se potrivească cu una din expresiile scrise în clauza GROUP BY. Funcția întoarce valoarea 0 sau 1.

Valorile întoarse de funcția GROUPING sunt folosite pentru a:

- Determina nivelul de totalizare pentru un anumit subtotal, adică grupul sau grupurile pe care se bazează subtotalul;
- Identifică dacă o valoare de NULL din coloana unui rând dintr-un set de rezultate indică:
 - valoare de NULL din tabela de baza (NULL stocat); - valoare de NULL creată de ROLLUP/CUBE (ca rezultat a unei funcții de grup asupra acelei expresii).

Valoare 0 returnată de funcția GROUPING bazată pe o expresie indică următoarele:

- Expresia a intrat în calcul unei valori totale.
- Valoarea NULL din expresia coloanei este o valoare stocată de NULL.

Valoare 1 returnată de funcția GROUPING bazată pe o expresie indică următoarele:

- Expresia nu a intrat în calcul unei valori totale.
- Valoarea NULL din expresia coloanei este o valoare creată de ROLLUP sau CUBE ca rezultat al grupării.



EXEMPLU:

```
SELECT department_id DEPTID, job_id JOB, SUM(salary),
FROM employees
WHERE department_id < 50
GROUP BY ROLLUP(department_id, job_id);
GROUPING(department_id) GRP_DEPT, GROUPING(job_id)
GRP_JOB
```

DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
10	AD_ASST	4400	0	0
10		4400	0	1
20	MK_MAN	13000	0	0
20	MK_REP	6000	0	0
20		19000	0	1
		23400	1	1

6 rows selected.

În exemplul de mai sus avem în primul rând valoarea însumată a salariilor și anume 4400. Această valoare este salariul total pentru job_ID='AD_ASST' din departamentul 10. Pentru a calcula această valoare totală, trebuie luate în considerație atât coloana DEPARTMENT_ID cât și JOB_ID. Astfel, valoare 0 este întoarsă atât pentru expresia GROUPING(department_id) cât și pentru GROUPING(job_id).

Luam acum în discuție valoarea totală 4400 din cel de-al doilea rând. Această valoare este salariul total pentru departamentul 10 și a fost calculată luând în considerare coloana DEPARTMENT_ID; astfel valoarea 0 a fost returnată de GROUPING(department_id). Deoarece coloana JOB_ID nu a fost luată în considerare în calculul acestei valori, GROUPING(job_id) returnează valoarea 1.

Rezultate similare se pot observa în rândul al cincilea.

Observați valoare 23400 din ultimul rând ce reprezintă salariul total pentru toate funcțiile și pentru acele departamente a căror număr este mai mic de 50. Pentru a calcula această valoare totală nu au fost avute în vedere nici coloana DEPARTMENT_ID nici JOB_ID. Astfel este returnată valoarea 1 pentru expresia GROUPING(department_id) și pentru GROUPING(job_id).

9.14 GROUPING SETS

GROUPING SETS reprezintă o extensie a clauzei **GROUP BY** care permite specificarea unei grupări multiple a datelor favorizând astfel o sumare eficientă și astfel facilitând analiza datelor la folosirea mai multe dimensiuni.

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

Serverul Oracle calculează toate grupările specificate în clauza **GROUPING SETS** și combină rezultatele grupurilor individuale cu o operație de tip **UNION ALL**.

Eficiența Grouping set este datorată :

- este necesară doar o singură citire a datelor din tabelul de baza;
- nu este necesară scrierea unei expresii complexe de tip UNION ;
- cu cât sunt mai multe elemente în GROUPING SETS cu atât crește beneficiul performanței dat de instrucțiune .

Pentru a specifica diverse tipuri de grupări (care pot include operatori de tip **ROLLUP** sau **CUBE**) este necesară doar o singură instrucțiune **SELECT** care poate fi scrisă împreună cu **GROUPING SETS** în locul unor fraze **SELECT** multiple combinate cu operatorul **UNION ALL**.

De exemplu se poate scrie:



```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY
GROUPING SETS
    ((department_id, job_id, manager_id),
     (department_id, manager_id),(job_id, manager_id));
```

Instrucțiunea calculează suma pentru următoarele grupuri : (department_id, job_id, manager_id), (department_id, manager_id) și (job_id, manager_id).

În lipsa acestei facilități oferite de Oracle9i ar fi trebuit scrise mai multe instrucțiuni SELECT combinate cu operatorul UNION ALL O abordare care folosește multe interogări este ineficientă deoarece necesită multiple citiri ale datelor. Comparați instrucțiunea precedentă cu alternativa :

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY CUBE(department_id, job_id, manager_id);
```

Instrucțiunea precedentă calculează toate cele 8 ($2^3 \cdot 2$) grupuri doar prin unica grupare : (department_id, job_id, manager_id), (department_id, manager_id) și (job_id, manager_id) care interesează.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

O altă alternativă este instrucțiunea următoare:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id, manager_id
UNION ALL
SELECT department_id, NULL, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, manager_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY job_id, manager_id;
```

Această instrucțiune necesită trei citiri ale tabelei de bază și este deci ineficientă. CUBE și ROLLUP pot fi gândiți ca grouping sets cu o semantică specifică (echivalentă din tabelul de mai jos)

CUBE(a, b, c) is equivalent to	GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())
ROLLUP(a, b, c) is equivalent to	GROUPING SETS ((a, b, c), (a, b), (a), ())

Exemplu



```
SELECT department_id, job_id, manager_id, avg(salary)
FROM employees
GROUP BY GROUPING SETS
((department_id, job_id), (job_id, manager_id));
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
10	AD_ASST		4400
20	MK_MAN		13000
20	MK_REP		6000
50	ST_CLERK		2925
50	ST_MAN		5800

← 1

11..				
	MK_MAN	100	13000	
	MK_REP	201	6000	
	SA_MAN	100	10500	
	SA_REP	149	8866.666667	
	ST_CLERK	124	2925	
	ST_MAN	100	5800	

← 2

26 rows selected.

Interogarea din exemplu calculează suma pentru două grupuri. Tabelul este împărțit în următoarele grupuri:

- Department_ID, Job_ID;
- Job_ID, Manager_ID.

Este calculată media salariilor pentru fiecare dintre aceste grupuri, iar seturile de rezultate sunt afișate pentru fiecare grup.

În lista rezultatelor, grupul marcat cu 1 poate fi interpretat astfel:

- Media salariilor pentru toți angajații ce au job_ID='AD_ASST' din departamentul 10 este 4400.
- Media salariilor pentru toți angajații ce au job_ID='MK_MAN' din departamentul 20 este 13000.
- Media salariilor pentru toți angajații ce au job_ID=' MK_REP' din departamentul 20 este 6000.
- Media salariilor pentru toți angajații ce au job_ID='ST_CLERK' din departamentul 50 este 2925 și aşa mai departe

În lista rezultatelor, grupul marcat cu 2 poate fi interpretat astfel:

- Media salariilor pentru toți angajații ce au job_ID='MK_MAN', ca și persoana cu manager_ID=100 este 13000.
- Media salariilor pentru toți angajații ce au job_ID='MK_REP', ca și persoana cu manager_ID=201 este 6000 și aşa mai departe.

Exemplul de mai sus poate fi scris după cum urmează :

```
SELECT department_id, job_id, NULL as manager_id,
AVG(salary) as AVGSAL FROM employees
GROUP BY department_id, job_id UNION ALL
SELECT NULL, job_id, manager_id, avg(salary) as AVGSAL
FROM employees GROUP BY job_id, manager_id;
```

În lipsa unui element de optimizare care să caute în blocurile interogării pentru a genera un plan de execuție, interogarea de mai sus necesită scanarea dublă a tabelei EMPLOYEES, lucru ce este foarte ineficient, motiv pentru care se recomandă folosirea instrucțiunii **GROUPING SETS**.

9.15 Composite Columns

O coloană compusă este o colecție de coloane care sunt tratate unitar în momentul efectuării calculelor pentru grupuri.

Pentru a specifica o coloană compusă în clauza **GROUP BY** trebuie să scrieți aceste coloane în paranteze astfel încât severul Oracle să poată recunoaște aceste coloane ca o unitate atunci când execută calculele pentru cluzele **ROLLUP** sau **CUBE**, de exemplu: **ROLLUP (a, (b, c), d)** unde (b,c) formează o coloană compusă care este tratată unitar.

În general folosirea coloanelor compuse este utilă la scrierea instrucțiunilor **ROLLUP**, **CUBE** sau **GROUPING SETS**. Dacă folosiți coloane compuse pentru instrucțiunile **ROLLUP** sau **CUBE** înseamnă ca dorîți să treceți peste anumite nivele de însumare **(b,c)**.

De exemplu **GROUP BY ROLLUP (a, (b, c))** este echivalent cu:

GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ()

Aici (b, c) sunt tratate ca o unitate iar rollup nu va fi aplicat pentru (b, c). Este că și cum ați folosi un alias, de exemplu z, pentru (b, c), și expresia GROUP BY se reduce la GROUP BY ROLLUP(a, z).



NOTĂ:

GROUP BY() este de obicei o cluză tipică pentru instrucțiunea SELECT cu valori de null pentru a și b și doar o funcție de grup. Se folosește în special pentru calcularea unui total general.



**SELECT NULL, NULL, aggregate_col
FROM <table_name>
GROUP BY ();**

Comparați sintaxa de forma, **GROUP BY ROLLUP(a, b, c)** cu :

**GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ().**

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Similar, **GROUP BY CUBE((a, b), c)** ar fi echivalent cu :

GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP By ()

Tabelul următor ilustrează diferența dintre folosirea coloanelor compuse și specificațiile pentru GROUP BY .

GROUPING SETS Statements	Equivalent GROUP BY Statements
GROUP BY GROUPING SETS(a, b, c)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
GROUP BY GROUPING SETS(a, b, (b, c)) (The GROUPING SETS expression has a composite column)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
GROUP BY GROUPING SETS((a, b, c))	GROUP BY a, b, c
GROUP BY GROUPING SETS(a, (b,))	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY ()
GROUP BY GROUPING SETS (a, ROLLUP(b, c)) (The GROUPING SETS expression has a composite column)	GROUP BY a UNION ALL GROUP BY ROLLUP(b, c)

EXEMPLU:



```
SELECT department_id, job_id, manager_id, SUM(salary)  
FROM employees  
GROUP BY ROLLUP( department_id,(job_id, manager_id));
```

Considerăm exemplul:



```
SELECT department_id, job_id,manager_id, SUM(salary)  
FROM employees  
GROUP BY ROLLUP( department_id,job_id, manager_id);
```

Interogarea presupune calcularea de către serverul Oracle Server a valorilor pentru următoarelor grupuri :

1. (department_id, job_id, manager_id);
2. (department_id, job_id);
3. (department_id);
4. ().

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Dacă sunteți interesați în gruparea după liniile (1), (3) și (4) din exemplul de mai sus, nu puteți limita calculul acestor grupuri fără a folosi coloane compuse.

Folosirea coloanelor compuse dă posibilitatea de a trata coloanele JOB_ID și MANAGER_ID unitar atunci când se executa ROLLUP. Coloanele incluse în paranteze sunt tratate unitar la execuția instrucțiunilor ROLLUP și CUBE, lucru ilustrat în figură.

Scriind coloanele JOB_ID și MANAGER_ID în paranteze, indicam serverului Oracle faptul că trebuie să trateze coloanele JOB_ID și MANAGER_ID unitar, drept coloană compusă.

Exemplul calculează grupurile :

- (department_id, job_id, manager_id);
- (department_id);
- () .

și afișează următoarele :

The diagram illustrates the execution of a query with ROLLUP. It shows two result sets. The top result set has three levels of grouping: department_id (1), department_id and job_id (2), and department_id, job_id, and manager_id (3). The bottom result set shows the total salary for all employees.

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
10	AD_ASST	101	4400
10			4400
20	MK_MAN	100	13000
20	MK_REP	201	6000
20			19000
50	ST_CLERK	124	11700
50	ST_MAN	100	5800
50			17500

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
10	AC_MGR		20300
10	SA_REP	149	7000
			7000
			17500

23 rows selected.

- Salariul total pe fiecare departament (eticheta 1)
- Salariul total pe fiecare departament, job_ID și manager (eticheta 2)
- Totalul general (eticheta 3)

Exemplul de mai sus poate fi scris și astfel :

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM employees GROUP BY department_id, job_id, manager_id
UNION ALL
SELECT department_id, TO_CHAR(NULL), TO_NUMBER(NULL),
SUM(salary) FROM employees GROUP BY department_id
UNION ALL
```

```
SELECT TO_NUMBER(NULL),  
TO_CHAR(NULL),TO_NUMBER(NULL),SUM(salary)  
FROM employees GROUP BY();
```

În absenta unui element de optimizare care să studieze interogarea și să creeze un plan de execuție a interogării, interogarea de mai sus necesită trei scanări ale tabelei de baza, EMPLOYEES, lucru ce este ineficient. De aceea se recomandă folosirea coloanelor compuse.

9.16 Concatenated Groupings

Grupările concatenate oferă o soluție concisă de a genera combinații utile de grupuri.

Pentru a specifica seturi de grupări concatenate trebuie să separați setările de grupuri multiple și operațiile de tip ROLLUP și CUBE prin virgule astfel încât serverul Oracle să le combine într-o singura clauza GROUP BY.

Rezultatul este intersecția valorilor grupurilor din fiecare set de grupări.

EXEMPLU

GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)

Exemplul precedent definește următoarele grupuri: (a, c), (a, d), (b, c), (b, d)

Concatenarea seturilor de grupări este foarte folositoare deoarece :

- ușurează scrierea instrucțiunii: nu trebuie să enumerați, prin scriere manuală, toate grupurile ;
- folosirea aplicației: SQL-ul general de aplicații OLAP include deseori concatenarea seturilor de grupare, pentru fiecare set de grupare definind seturile de grupare necesare pentru dimensionare.

EXEMPLU:

```
SELECT department_id, job_id, manager_id, SUM(salary)  
FROM employees GROUP BY  
department_id,ROLLUP(job_id),CUBE(manager_id);
```

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)	
10	AD_ASST	101	4400	← ①
20	MK_MAN	100	13000	
20	MK_REP	201	6000	
50	ST_MAN	100	5800	
10		101	4400	← ②
20		201	6000	
10	AD_ASST		4400	← ③
10			4400	← ④

49 rows selected.

Din exemplul de mai sus rezultă formarea următoarelor grupuri :

- (department_id, manager_id, job_id);
- (department_id, manager_id);
- (department_id, job_id);
- (department_id).

Se calculează salariul total pentru fiecare grup, exemplul afișând următoarele :

- Salariul total pentru fiecare department, job ID, manager (eticheta 1);
- Salariul total pentru fiecare department, manager ID (eticheta 2);
- Salariul total pentru fiecare department, job ID (eticheta 3);
- Salariul total pentru fiecare department (eticheta 4).

Pentru o înțelegere mai ușoară, detaliile pentru departamentul 10 sunt evidențiate în figura de mai sus.



9.17 Exerciții

1. Funcțiile de grup acționează asupra mai multor rânduri și produc un rezultat. (adevărat/fals).
2. Funcțiile de grup includ valoarea null în calcule (adevărat/fals).
3. Clauza WHERE restricționează rândurile anterior incluziei acestora în grupurile de calcul (adevărat/fals).
4. Afipați cel mai mare salar, cel mai mic salar, suma și media salariului pentru toți angajații. Etichetați coloanele cu Maxim, Minim, Suma și Media. Rotunjiți rezultatele (fără zecimale). Salvați instrucțiunea în fișierul p4.sql.

Maximum	Minimum	Sum	Average
24000	2500	175500	8775

5. Modificați p4.sql astfel încât să afișeze aceleași informații pentru fiecare tip de meserie. Salvați modificările în p5.sql.

JOB_ID	Maximum	Minimum	Sum	Average
AC_ACCOUNT	8300	8300	8300	8300
AC_MGR	12000	12000	12000	12000
AD_ASST	4400	4400	4400	4400
AD_PRES	24000	24000	24000	24000
AD_VP	17000	17000	34000	17000
IT_PROG	9000	4200	19200	6400
MK_MAN	13000	13000	13000	13000
MK_REP	6000	6000	6000	6000
SA_MAN	10500	10500	10500	10500
SA_REP	11000	7000	26600	8867
ST_CLERK	3500	2500	11700	2925
ST_MAN	5800	5800	5800	5800

12 rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

6. Scrieți o interogare pentru afișarea numărului de angajai cu aceeași meserie.

JOB_ID	COUNT()
AC_ACCOUNT	1
AC_MGR	1
AD_ASST	1
AD_PRES	1
AD_VP	2
IT_PROG	3
MK_MAN	1
MK_REP	1
SA_MAN	1
SA_REP	3
ST_CLERK	4
ST_MAN	1

12 rows selected.

7. Determinați numărul managerilor fără să-i listați (doar numărul lor). Etichetați coloana Nr. Manageri.(Folosiți coloana Manager_ID).

Number of Managers
8

8. Scrieți o interogare care să afișeze diferența dintre salariile cele mai mari și cele mai mici. Etichetați coloana Diferență.

DIFFERENCE
21500

9. Afipați numărul managerului și salariul celui mai prost plătit angajat pentru acel manager. Excludeți pe cei care nu au manager. Excludeți grupurile care au salariul minim mai mic decât 1600\$. Sortați rezultatele în ordine descrescătoare după salar.

MANAGER_ID	MIN(SALARY)
102	9000
205	8300
149	7000

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

10. Scrieți o interogare care să afișeze numele departamentului, localitatea, numărul de angajai și salariul mediu al angajaților din acel departament. Etichetați coloanele Dname, Loc, No of People și Salary. Vezi exemplul.

Name	Location	Number of People	Salary
Accounting	1700	2	10150
Administration	1700	1	4400
Executive	1700	3	19333.33
IT	1400	3	6400
Marketing	1800	2	9500
Sales	2500	3	10033.33
Shipping	1500	5	3500

7 rows selected.

11. Scrieți o interogare care afișează numărul total de angajai și numărul total de angajați care au fost angajați în anii 1995, 1996, 1997 și 1998. Etichetați coloanele corespunzător. Vezi exemplul.

TOTAL	1995	1996	1997	1998
20	1	2	2	3

12. Afipați meseria, suma salariilor pentru meseria respectivă din cadrul departamentelor 20, 50, 80, 90 precum și salariul total pentru acea meserie pentru toate departamentele. Etichetați coloanele corespunzător. Vezi exemplul.

Job	Dept 20	Dept 50	Dept 80	Dept 90	Total
AC_ACCOUNT					8300
AC_MGR					12000
AD_ASST					4400
AD_PRES				24000	24000
AD_VP				34000	34000
IT_PROG					19200
MK_MAN	13000				13000
MK_REP	6000				6000
SA_MAN			10500		10500
SA_REP			19600		26600
ST_CLERK		11700			11700
ST_MAN		5800			5800

12 rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

13. Scrieți o interogare care să afișeze următoarele informații pentru acei angajați a căror manager_ID este mai mic de 120:

- Manager_ID;
- Job_ID și salariul total pentru fiecare job_ID pentru angajații care au același sef;
- Salariul total pe care șefii trebuie să-l plătească angajaților ;
- Salariul total pe care șefii trebuie să-l plătească angajaților, indiferent de job_ID.

MANAGER_ID	JOB_ID	SUM(SALARY)
100	AD_VP	34000
100	MK_MAN	13000
100	SA_MAN	10500
100	ST_MAN	5800
100		63300
101	AC_MGR	12000
101	AD_ASST	4400
101		16400
102	IT_PROG	9000
102		9000
103	IT_PROG	10200
103		10200
		98900

13 rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

14. Observați rezultatele de la punctul 13. Scrieți o interogare folosind funcții de grupare pentru a determina dacă valorile de NULL din coloanele corespunzătoare expresiilor GROUP BY sunt cauzate de operația ROLLUP.

MGR	JOB	SUM(SALARY)	GROUPING(MANAGER_ID)	GROUPING(JOB_ID)
100	AD_VP	34000	0	0
100	MK_MAN	13000	0	0
100	SA_MAN	10500	0	0
100	ST_MAN	5800	0	0
100		63300	0	1
101	AC_MGR	12000	0	0
101	AD_ASST	4400	0	0
101		16400	0	1
102	IT_PROG	9000	0	0
102		9000	0	1
103	IT_PROG	10200	0	0
103		10200	0	1
		98900	1	1

13 rows selected.

MANAGER_ID	JOB_ID	SUM(SALARY)
100	AD_VP	34000
100	MK_MAN	13000
100	SA_MAN	10500
100	ST_MAN	5800
100		63300
101	AC_MGR	12000
101	AD_ASST	4400
101		16400
102	IT_PROG	9000
102		9000
103	IT_PROG	10200
103		10200
	AC_MGR	12000
	AD_ASST	4400
	AD_VP	34000
	IT_PROG	19200
	MK_MAN	13000
	SA_MAN	10500
	ST_MAN	5800
		98900

20 rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

15. Scrieți o interogare care să afișeze următoarele informații despre angajații a căror manager_ID este mai mic de 120 :

- Manager_ID;
- Funcția și salariul total pentru fiecare funcție pentru angajații care au același sef;
- Salariul total al angajaților ce au același sef ;
- Salariul total pe fiecare funcție, indiferent de sef;
- Salariul total indiferent de funcții.

16. Observați rezultatele interogării de la punctul 15. Scrieți o interogare care să folosească funcțiile de grupare pentru a determina dacă valorile de NULL din coloanele corespunzătoare clauzei GROUP BY sunt date de operatorul CUBE.

MGR	JOB	SUM(SALARY)	GROUPING(MANAGER_ID)	GROUPING(JOB_ID)
100	AD_VP	34000	0	0
100	MK_MAN	13000	0	0
100	SA_MAN	10500	0	0
100	ST_MAN	5800	0	0
100		63300	0	1
101	AC_MGR	12000	0	0
101	AD_ASST	4400	0	0
101		16400	0	1
102	IT_PROG	9000	0	0
102		9000	0	1
103	IT_PROG	10200	0	0
103		10200	0	1
	AC_MGR	12000	1	0
	AD_ASST	4400	1	0
	AD_VP	34000	1	0
	IT_PROG	19200	1	0
	MK_MAN	13000	1	0
	SA_MAN	10500	1	0
	ST_MAN	5800	1	0
		98900	1	1

20 rows selected.

17. Scrieți o interogare, folosind GROUPING SETS, pentru a afișa următoarele grupuri:

- department_id, manager_id, job_id;
- department_id, job_id;
- manager_id, job_id.

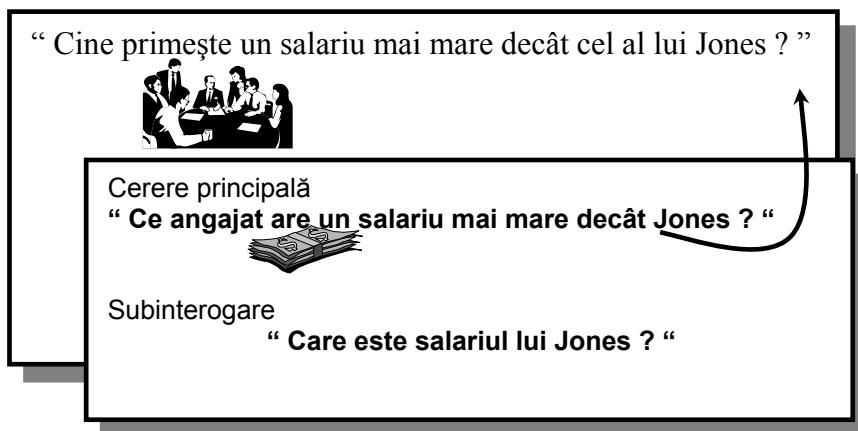
Interogarea trebuie să calculeze suma salariilor pentru fiecare dintre aceste grupuri.

Capitolul 10 Subinterrogari

Obiective:

- Descrierea tipurilor de probleme pe care le pot rezolva subinterrogările;
- Definirea subinterrogărilor;
- Enumerarea tipurilor de subinterrogări;
- Scrierea subinterrogărilor de un singur rând și a celor de mai multe rânduri.

În acest capitol se vor studia caracteristici mai avansate ale instrucțiunii SELECT cum ar fi scrierea subinterrogărilor în clauza WHERE a altrei instrucțiuni SQL în scopul obținerii de valori bazate pe o valoare necunoscută a unei condiții.



10.1 Folosirea unei subinterrogări pentru a rezolva o problema

Să presupunem că se dorește scrierea unei interogări pentru a afla cine câștigă un salariu mai mare decât salariul lui Jones.

Pentru rezolvarea acestei probleme este nevoie de două interogări: o interogare pentru a afla ce salariu câștigă Jones și o două pentru a determina cine câștigă mai mult decât această sumă.

Problema poate fi rezolvată combinând aceste două interogări, integrând una din cereri în cealaltă.

O cerere inclusă sau o subinterrogare returnează o valoare care va fi folosită de către interroarea exterioară sau principală. Folosirea unei subinterrogări este echivalentă executării a două cereri secvențiale și folosirii rezultatului primei cereri ca valoare de căutare pentru cea de a două cerere.



```
SELECT select_list  
FROM table  
WHERE expr operator  
(SELECT select_list FROM table);
```



Subinterrogarea (cererea internă) se execută o singură dată, înaintea interrogației principale. Rezultatul subinterrogării este utilizat de către cererea principală (cererea externă).

O subinterrogare reprezintă o instrucțiune SELECT care este inclusă într-o clauză aparținând altrei instrucțiuni SELECT. Prin utilizarea subinterrogărilor se pot construi instrucțiuni mai puternice pornind de la instrucțiuni simple. Acestea pot fi foarte folositoare în cazurile în care se dorește selectarea unor rânduri dintr-un tabel folosind o condiție care depinde de datele din tabelul propriu-zis.

Subinterrogările pot fi plasate în următoarele clauze SQL:

- **WHERE**
- **HAVING**
- **FROM**

În sintaxa mai sus prezentată expr operator implică unul din următorii operatori de comparație: $>$, $=$ sau IN.



Operatorii de comparare se împart în două clase:

- operatori pentru subinterrogări de un singur rând: \geq , $<$, \neq , \leq
- operatori pentru subinterrogări de mai multe rânduri: IN, ANY, ALL.

Subinterrogarea este deseori referită ca fiind o instrucțiune SELECT inclusă, sub-SELECT sau instrucțiune SELECT internă. În general, subinterrogarea se execută prima, iar rezultatul este folosit pentru a finaliza condiția de cerere pentru interroarea principală sau externă.



```
SELECT last_name  
FROM employees  
WHERE salary >  
(SELECT salary  
FROM employees  
WHERE last_name = 'Abel');
```

LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

În figura anterioară, cererea internă determină salariul angajatului cu numele „Abel”. Cererea externă preia rezultatul cererii interne și îl folosește pentru a afișa toți angajații care au salariul mai mare decât această sumă.



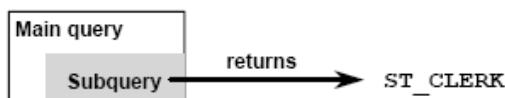
Reguli în utilizarea subinterrogărilor

- O subinterrogare trebuie să fie inclusă între paranteze.
- O subinterrogare trebuie să apară în partea dreapta a unui operator de comparare.
- Subinterrogările nu pot conține clauza ORDER BY. Pentru o instrucțiune SELECT poate exista doar o singură clauză ORDER BY, iar dacă această clauza este specificată, ea trebuie să fie ultima clauză din instrucțiunea SELECT principală.

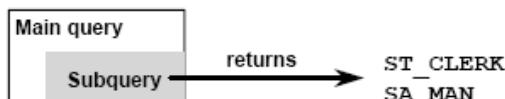
Subinterrogările folosesc două clase de operatori de comparare: operatori single-row și operatori multiple-row.

10.2 Tipuri de subinterrogări

- Single-row subquery



- Multiple-row subquery



Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

- Single-row : cereri care returnează doar un rând din instrucțiunea SELECT internă;
- Multiple-row : cereri care returnează mai mult de un rând din instrucțiunea SELECT internă.

10.3 Subinterrogari single-row

O subinterrogare single-row este acea subinterrogare care returnează un singur rând din instrucțiunea SELECT internă. Acest tip de subinterrogare folosește un operator single-row.



Exemplu:

```
SELECT last_name, job_id  
FROM employees  
WHERE job_id =  
    (SELECT job_id  
     FROM employees  
     WHERE employee_id = 141);
```

LAST_NAME	JOB_ID
Rajs	ST_CLERK
Davies	ST_CLERK
Matos	ST_CLERK
Vargas	ST_CLERK

Afișarea angajaților care lucrează pe același post (au aceeași meserie) ca și angajatul care are numărul de marcă 141.

Executarea unei subinterrogări single-row:

```
SELECT last_name, job_id, salary  
FROM   employees  
WHERE  job_id = ST_CLERK  
       (SELECT job_id  
        FROM   employees  
        WHERE  employee_id = 141)  
AND    salary > 2600  
       (SELECT salary  
        FROM   employees  
        WHERE  employee_id = 143);
```

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3600
Davies	ST_CLERK	3100

Facultatea de Automatică și Calculatoare Iași ***Baze de date – lucrări practice***

O instrucțiune SELECT poate fi considerată ca un bloc de cereri. Exemplul de mai sus afișează angajații a căror funcție este aceeași cu cea a angajatului cu numărul 141 și a căror salar este mai mare decât cel al angajatului 143.

Exemplul este format din 3 blocuri de cereri: o cerere exterioară și două cereri interne. Blocurile de cereri interne sunt executate primele, producând rezultatele: FUNCTIONAR (ST_CLERK), respectiv 2600. Apoi este procesat blocul de cereri exterior care folosește valorile returnate de către cererile interne pentru a finaliza propriile condiții de căutare.

Ambele cereri interne returnează valori singulare (FUNCTIONAR și 2600), astfel ca această instrucțiune SQL este denumită subinterrogare single-row.



Interogările exterioare și incluse pot prelua datele din tabele diferite.



10.4 Utilizarea funcțiilor de grup într-o subinterrogare

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE salary = (SELECT MIN(salary) FROM employees);
```

LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

În interogarea principală pot fi afișate date prin utilizarea unei funcții de grup folosind o subinterrogare care să returneze un singur rând. Subinterrogarea se va plasa între paranteze, după operatorul de comparare.

Exemplul din figura precedentă afișează numele, funcția și salariul tuturor angajaților al căror salar este egal cu salariul minim. Funcția **MIN** (funcție de grup) returnează o singură valoare (și anume 2500), care este folosită de către interogarea principală.

Clauza HAVING în subinterrogare

Server-ul Oracle execută mai întâi subinterrogările. Server-ul Oracle returnează rezultatele către clauza **HAVING** a interogării principale.



```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) >
       (SELECT MIN(salary)
        FROM employees
        WHERE department_id = 50);
```

DEPARTMENT_ID	MIN(SALARY)
10	4400
20	4900

7 rows selected.

Subinterogările pot fi folosite nu numai în clauza **WHERE** ci și în clauza **HAVING**. Server-ul Oracle execută subinterrogarea returnând rezultatul către clauza **HAVING** a subinterrogării principale.

Instrucțiunea SQL prezentată în figura de mai sus are ca scop final afișarea tuturor departamentelor la nivelul cărora salariul minim are o valoare mai mare decât valoarea salariului minim din cadrul departamentului 50.



Exemplu: Se cere să se găsească funcția având cel mai scăzut salariu mediu.

```
SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
                      FROM employees
                      GROUP BY job_id);
```

10.5 Erori ce pot apărea la folosirea subinterrogărilor



```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
       (SELECT MIN(salary)
        FROM   employees
        GROUP BY department_id);
```

Single-row operator with multiple-row subquery
ERROR at line 4
ORA-01427: single-row subquery returns more than one row

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

O eroare obișnuită la folosirea subinterrogărilor o reprezintă returnarea a mai mult de un rând de către o subinterrogare dorită a fi de tip single-row.

În instrucțiunea SQL din exemplul anterior, subinterrogarea conține o clauză GROUP BY după numărul departamentului (department_ID), care implică selectarea mai multor rânduri, câte unul pentru fiecare grup găsit. În acest caz, rezultatul subinterrogării va fi: 4400, 6000, 2500, 4200, 7000, 17000 și 8300.

Interrogarea externă preia rezultatele subinterrogării și le folosește în clauza WHERE. Clauza WHERE conține operatorul egal (=), operator de comparare single-row, care așteaptă o singură valoare în partea sa dreaptă. Operatorul ‘ = ’ nu poate accepta mai mult de o valoare primită de la subinterrogare și astfel este generată eroarea.

Pentru a corecta eroarea, operatorul (=) trebuie înlocuit cu operatorul IN.

```
SELECT last_name, job_id
FROM employees
WHERE job_id =
      (SELECT job_id
       FROM employees
       WHERE last_name='Haas');
```

Subquery returns no values

no rows selected

O problemă obișnuită legată de subinterrogări o constituie posibilitatea neselectării nici unui rând de către interrogarea inclusă.

În ceea ce privește instrucțiunea SQL de mai sus, subinterrogarea conține o clauză WHERE (ename = 'Haas'). Se presupune că intenția este de a selecta angajatul cu numele Haas. Instrucțiunea pare a fi corectă, dar la execuție nu se selectează nici un rând.

Problema este ortografierea greșită a cuvântului Haas. Nu există nici un angajat cu numele de Haas. Astfel, subinterrogarea nu va selecta nici un rând. Interrogarea externă preia rezultatul subinterrogării (null, în acest caz) și folosește acest rezultat în propria-i clauza WHERE. Interrogarea externă nu găsește nici un angajat având câmpul referitor la funcție de valoare nulă și astfel nu returnează nici un rând.

10.6 Subinterrogari multiple-row

- Selectează mai mult de un rând
- Folosesc operatori multiple-row de comparare

Operator	Semnificație
IN	Egal cu oricare din elementele listei
ANY	Compară valoarea cu fiecare valoare returnată de subinterrogare luată separat
ALL	Compară valoarea cu toate valorile returnate de subinterrogare

Subinterrogările care returnează mai mult de un rând se numesc subinterrogări multiple-row. În cazul subinterrogărilor multiple-row se folosesc operatori multiple-row în locul celor single-row. Operatorul multiple-row necesită una sau mai multe valori.



```
SELECT last_name, salary, department_id  
FROM employees  
WHERE salary IN (SELECT MIN(salary)  
FROM employees  
GROUP BY department_id);
```

Se cere să se selecteze angajații care câștigă un salariu egal cu salariul minim la nivel de departament.

Interrogarea internă va fi prima executată producând un răspuns. Blocul cererii externe este apoi procesat și se folosesc valorile returnate de către interrogarea inclusă pentru finalizarea propriei condiții de căutare. De fapt, interrogarea principală este privită din perspectiva server-ului Oracle astfel:

```
SELECT last_name, salary, department_id  
FROM employees  
WHERE salary IN (2500, 4200, 4400, 6000, 7000, 8300, 8600, 17000);
```

10.7 Utilizarea operatorului ANY în subinterogările multiple-row



```
SELECT employee_id, last_name, job_id, salary
FROM   employees      9000, 6000, 4200
WHERE  salary < ANY
       (SELECT salary
        FROM   employees
        WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
206	Gietz	SA_MGR	4300

10 rows selected.

Operatorul ANY compară o valoare cu fiecare valoare returnată de subinterrogare. Exemplul de mai sus afișează angajații ale căror salarii sunt mai mici decât al oricărui programator (IT_PROG) dar care nu sunt programatori. Salariul maxim pe care îl câștigă un programator este \$9000. Instrucțiunea SQL afișează toți angajații care nu sunt programatori dar câștigă mai puțin de \$9000.



- < ANY → înseamnă mai mic decât maxim.
- > ANY → înseamnă mai mare decât minim.
- = ANY → este echivalent cu operatorul IN.

10.8 Utilizarea operatorului ALL în subinterogările multiple-row

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
       9000, 6000, 4200
       (SELECT salary
        FROM   employees
        WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

Operatorul **ALL** compară o valoare cu toate valorile returnate de o subinterrogare. Exemplul de mai sus afișează toți angajații ale căror salarii sunt mai mici decât al oricărui programator (IT_PROG) dar care nu sunt programatori. Cel mai mic salar al vreunui programator este \$4200, așa că interrogarea va selecta acei angajați ale căror salarii sunt mai mici decât \$4200.



> ALL → înseamnă mai mare decât maxim
< ALL → înseamnă mai mic decât minim

Operatorul **NOT** poate fi folosit împreună cu operatorii **IN**, **ANY** și **ALL**.

10.9 Returnarea valorilor nule în rezultatul subinterrogării



```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id NOT IN
    (SELECT mgr.manager_id FROM employees mgr);
```

Comanda SQL de mai sus afișează toți angajații care nu au nici un subordonat. Logic, acesta interogare SQL ar trebui să returneze 12 linii dar nu se returnează nici una. Una din valorile returnate de interogare este o valoare de null și prin urmare întreaga interogare nu returnează nici o linie.

Motivul este că toate aceste condiții care compară rezultatul unei valori cu null transformă rezultatul în null. **Când există valori de null în rezultatul subinterrogării, nu folosiți operatorul NOT IN.**

Operatorul **NOT IN** este echivalent cu **<> ALL**. Trebuie să aveți în vedere că valorile nulle ale rezultatului subinterrogării nu vor constitui o problemă dacă folosiți operatorul **IN**. Operatorul **IN** este echivalent cu operatorul **ANY**. De exemplu, pentru afișarea angajaților care nu au subordonați, folosiți următoarea exprimare SQL:

```
SELECT last_name FROM employees
WHERE employee_id NOT IN
    (SELECT manager_id FROM employees
     WHERE manager_id IS NOT NULL);
```

10.10 Subinterrogari de coloane multiple

Până acum am scris subinterrogări ce returnau una sau mai multe linii dar care comparau valorile pentru o singură coloană în clauza **WHERE** sau **HAVING** a instrucțiunii **SELECT**.

Dacă se dorește compararea uneia sau a mai multor coloane, la scrierea condițiilor pentru clauza **WHERE** trebuie folosiți operatorii logici.

Folosirea subinterrogărilor de coloane multiple oferă posibilitatea îmbinării condițiilor din două cluze **WHERE** în una singură.



```
SELECT column, column,...
FROM table
WHERE (column, column,...) IN
(SELECT column, column,...
FROM table
WHERE condition);
```



Afişați numele, numărul departamentului, salariul și comisionul oricărui angajat a cărui salariu și comision se potrivesc cu salariul și comisionul oricărui angajat din departamentul 30.

Exemplul de mai sus folosește o subinterrogare de coloane multiple (care returnează mai mult de o coloană) pentru a compara salariul și comisionul.

10.11 Compararea coloanelor (pereche și nepereche)

Compararea coloanelor într-o subinterrogare de coloane multiple poate fi făcută în două moduri: pereche sau nepereche.

În exemplul următor, clauza WHERE conține o comparație pereche. Fiecare linie returnată de comanda SELECT trebuie să aibă aceeași valoare atât în coloana MANAGER_ID cât și în coloana DEPARTMENT_ID cu cele ale angajatului cu numărul 178 sau 174.

În cazul unei comparații nepereche, fiecare valoare a coloanelor din clauza WHERE a interrogării principale va fi comparată în mod individual cu valorile multiple returnate de subinterrogare. Coloanele individuale pot să se potrivească cu orice valoare returnată de subinterrogare. Toate rândurile afișate trebuie să satisfacă, în mod colectiv, toate condițiile multiple din interrogarea principală. Un exemplu de comparație nepereche este dat mai jos.



Comparație pereche

```
SELECT employee_id, manager_id, department_id  
FROM employees  
WHERE (manager_id, department_id) IN  
(SELECT manager_id, department_id  
FROM employees  
WHERE employee_id IN (178,174))  
AND employee_id NOT IN (178,174);
```

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
176	149	80



Comparație nepereche

```
SELECT employee_id, manager_id, department_id  
FROM employees  
WHERE manager_id IN  
(SELECT manager_id  
FROM employees  
WHERE employee_id IN (174,141))  
AND department_id IN  
(SELECT department_id  
FROM employees  
WHERE employee_id IN (174,141))  
AND employee_id NOT IN(174,141);
```

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
142	124	50
143	124	50
144	124	50
176	149	80

10.12 Folosirea unei subinterrogări în clauza FROM

Se poate folosi o subinterrogare și în clauza FROM a instrucțiunii SELECT.

Exemplul următor afișează numele angajaților, salariul, numărul departamentului și media salariailor pentru toți angajații care câștigă mai mult

decât salariul mediu din departamentul în care lucrează.



Exemplu

```
SELECT a.last_name, a.salary, a.department_id, b.salavg
FROM employees a, (SELECT department_id,
AVG(salary) salavg
FROM employees
GROUP BY department_id) b
WHERE a.department_id = b.department_id
AND a.salary > b.salavg;
```

LAST_NAME	SALARY	DEPARTMENT_ID	SALAVG
Hartstein	13000	20	9500
Mourgos	5800	50	3500
Hunold	9000	60	6400
Zlotkey	10500	80	10033.3333
Abel	11000	80	10033.3333
King	24000	90	19333.3333
Higgins	12000	110	10150

7 rows selected.

10.13 Expresii scalare returnate de subinterrogări

O expresie scalară returnată de o subinterrogare este o subinterrogare care returnează valoarea exactă a unei coloane aferentă unui singur rând.

În Oracle8i subinterrogările scalare erau suportate doar în câteva cazuri similare cum ar fi:

- instrucțiunea **SELECT** (clauza FROM, WHERE);
- liste de valori din instrucțiunea **INSERT** .

În Oracle9i, subinterrogările scalare pot fi folosite în :

- Condiții și expresii ce fac parte din instrucțiuni ca DECODE și CASE;
- Toate clauzele SELECT cu excepția GROUP BY;
- În partea stângă a unui operator în clauza SET și WHERE a unei instrucțiuni UPDATE.

Subinterrogările de mai multe coloane scrise pentru a compara două sau mai multe coloane, în scrierea cărora se folosesc la clauza WHERE condiții compuse și operatori logici, nu sunt calificate ca subinterrogări scalare.

Valoarea expresiilor scalare ale subinterrogării este valoarea elementului returnat de subinterrogare. Dacă subinterrogarea nu întoarce nici un rând, valoarea expresiei scalare este NULL. Dacă subinterrogarea întoarce mai mult de un rând, serverul Oracle returnează o eroare.

Expresiile scalare date de subinterrogări nu sunt valide în următoarele cazuri:

- ca valori implicate pentru coloane și expresii pentru clustere;
- în clauza RETURNING a unei instrucțiuni DML ;
- ca bază a unei funcții de bază pentru index;
- în clauza GROUP BY, constrângeri CHECK, condiții pentru WHEN;
- clauza HAVING;
- în clauzele START WITH și CONNECT BY;
- în instrucțiuni care nu sunt în legătură cu interogări, cum ar fi CREATE PROFILE.



Exemplu

Scalar Subqueries in CASE Expressions

```
SELECT employee_id, last_name,
(CASE
    WHEN department_id = 20
        (SELECT department_id FROM departments
         WHERE location_id = 1800)
    THEN 'Canada' ELSE 'USA' END) location
FROM employees;
```

Scalar Subqueries in ORDER BY Clause

```
SELECT employee_id, last_name
FROM employees e
ORDER BY (SELECT department_name
          FROM departments d
          WHERE e.department_id = d.department_id);
```

Primul exemplu de mai sus demonstrează faptul că subinterrogările scalare pot fi folosite în expresii de tip CASE. Subinterrogarea returnează valoarea 20, care este ID-ul departamentului pentru care location ID este 1800.

Expresia CASE din interogarea principală folosește rezultatul subinterrogării pentru a afișa employee ID, last name, și valoarea Canada sau USA, în funcție de numărul departamentului, dacă acesta este sau nu 20.

Rezultatul exemplului este următorul :

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

EMPLOYEE_ID	LAST_NAME	LOCATI
100	King	USA
101	Kochhar	USA
102	De Haan	USA
103	Hunold	USA
104	Martie	USA
201	Hartstein	Canada
202	Fay	Canada
205	Higgins	USA
206	Gietz	USA

20 rows selected.

Cel de al doilea exemplu ordonează rezultatele în funcție de DEPARTMENT_NAME prin potrivirea valorilor pentru DEPARTMENT_ID din tabela EMPLOYEES cu valorile pentru DEPARTMENT_ID din tabela DEPARTMENTS. Această comparare este făcută prin folosirea subinterrogării scalare din clauza ORDER BY. Rezultatul este afișat mai jos.

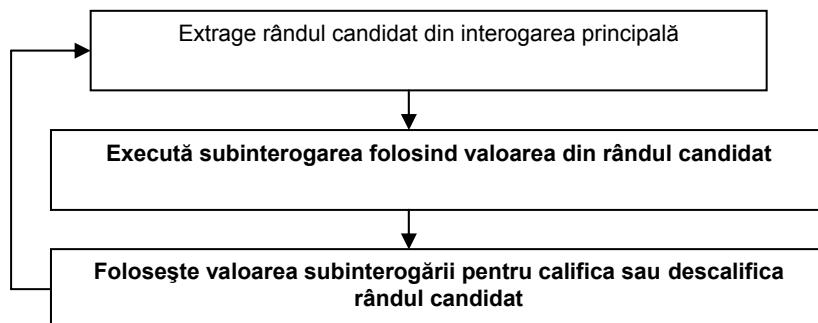
EMPLOYEE_ID	LAST_NAME
205	Higgins
206	Gietz
200	Whalen
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
..	...
142	Davies
143	Matos
144	Vargas
178	Grant

20 rows selected.

Acest exemplu folosește o subinterrogare corelată. Într-o subinterrogare corelată, subinterrogarea referă o coloană dintr-o tabelă referită în subinterrogarea principală.

10.14 Subinterrogări corelate

Subinterrogările corelate sunt folosite pentru procesare rând cu rând. Fiecare subinterrogare este executată odată pentru fiecare rând al interrogării principale.



Serverul Oracle execută o subinterrogare corelată atunci când o subinterrogare referă o coloană dintr-un tabel referit în interogarea principală. O subinterrogare corelată este evaluată de fiecare dată pentru fiecare rând procesat de interogarea principală. Interogarea principală poate fi o instrucție SELECT, UPDATE sau DELETE.

Subinterrogări imbricate versus subinterrogări corelate

Într-o subinterrogare imbricată normală, interogarea SELECT interioară este executată mai întâi odată, valoarea întoarsă fiind folosită de interogarea principală.

O subinterrogare corelată este executată odată pentru fiecare rând candidat dat de interogarea principală. Cu alte cuvinte, subinterrogarea este condusă de interogarea principală.

Executarea subinterrogării imbricate

- Se execută mai întâi subinterrogarea și este returnată o valoare;
- Se execută odată interogarea principală folosind valoarea dată de subinterrogare.

Executarea subinterrogării corelate

- Se ia un rând candidat, dat de interogarea principală;
- Se execută subinterrogarea folosind valoarea din rândul candidat;
- Se folosește valoarea dată de subinterrogare pentru a califica sau descalifica rândul candidat;
- Se repetă procedura până la epuizarea rândurilor candidat.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



```
SELECT column1, column2, ...
FROM table1
WHERE column1 operator
(SELECT column1, column2
FROM table2
WHERE expr1 = outer.expr2);
```



Operatorii **ANY** și **ALL** pot fi folosiți în subinterrogări corelate.



Exemplu: Găsiți toți angajații care câștigă mai mult decât salariul mediu din departamentul în care lucrează.

```
SELECT last_name, salary, department_id
FROM employees outer
WHERE salary >
(SELECT AVG(salary)
FROM employees
WHERE department_id = outer.department_id);
```

În acest caz subinterrogarea corelează salariul mediu din fiecare departament cu salariul fiecărui angajat. Deoarece atât interrogarea principală cât și subinterrogarea folosesc tabela EMPLOYEES în clauza FROM, acesteia î se atribuie un alias în interrogarea principală, nu doar pentru claritatea citirii ci și datorită faptului că fără acest alias interrogarea nu ar funcționa corect, deoarece subinterrogarea nu ar putea să facă diferență dintre coloana din interrogarea principală și cea din subinterrogare.



Exemplu: Afipați datele angajaților care și-au schimbat funcția de cel puțin două ori.

```
SELECT e.employee_id, last_name, e.job_id
FROM employees e
WHERE 2 <= (SELECT COUNT(*)
FROM job_history
WHERE employee_id = e.employee_id);
```

EMPLOYEE_ID	LAST_NAME	JOB_ID
101	Kochhar	AD_VP
176	Taylor	SA_REP
200	Whalen	AD_ASST

Serverul Oracle evaluează subinterrogarea corelată astfel:

- Selectează un rând din tabele specificată în interrogarea principală, adică un rând candidat.
- Memorează valoarea din rândul candidat pentru coloana referită în subinterrogare (în exemplu, coloana este E.EMPLOYEE_ID.)
- Execută subinterrogarea cu condiția dată de valoarea rândului candidat din interrogarea principală (în exemplu, funcția de grup COUNT(*) este evaluată pe baza valorii coloanei E.EMPLOYEE_ID obținută la punctul 2).
- Evaluează clauza WHERE a interrogării principale pe baza rezultatului dat de subinterrogare obținut în pasul 3. Acest lucru determină dacă rândul candidat va fi selectat pentru afișare sau nu. (în exemplu, numărul de schimbări ale funcției pentru un salariat, evaluat de subinterrogare este comparat cu acel 2 din clauza WHERE a interrogării principale. Dacă condiția este îndeplinită de acel angajat, rândul respectiv, este afișat.)
- Procedura se repetă pentru următoarele rânduri candidate până la finalul tabelei.

10.15 Folosirea operatorului EXISTS

Operatorul **EXISTS** verifică existența unei valori în setul de rezultate al unei subinterrogări.

Dacă subinterrogarea întoarce un rând atunci:

- Condiția de căutare nu mai continuă în interrogarea principală;
- Condiția este marcată ca fiind **TRUE**.

Dacă subinterrogarea nu întoarce niciun rând atunci:

- Condiția este marcată ca fiind **FALSE**;
- Condiția de căutare continuă în interrogarea principală.

În instrucțiunile SELECT imbricate sunt acceptați toți operatorii logici. Suplimentar se poate folosi operatorul EXISTS. Acest operator este folosit în mod frecvent în subinterrogările corelate pentru a testa dacă o valoare returnată de interrogarea principală există sau nu în setul de rezultate dat de interrogarea secundară. Dacă subinterrogarea întoarce cel puțin un rând, operatorul întoarce valoarea **TRUE**. Dacă valoarea nu există, întoarce **FALSE**. În acest sens operatorul NOT EXISTS testează dacă o valoare găsită de interrogarea principală este sau nu parte a setului de rezultate dat de subinterrogare.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



Exemplu: Găsiți angajatul care are cel puțin un subaltern.

```
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
      WHERE EXISTS ( SELECT 'X'
                      FROM employees
                     WHERE manager_id =
                           outer.employee_id);
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
124	Mourgos	ST_MAN	50
149	Zlotkey	SA_MAN	80
201	Hartstein	MK_MAN	20
205	Higgins	AC_MGR	110

8 rows selected.

Operatorul EXISTS asigură asupra faptului că regăsirea datelor în subinterrogare nu continuă atunci când s-a găsit cel puțin un rând care să satisfacă condiția: WHERE manager_id = outer.employee_id.

De notat faptul că subinterrogarea nu trebuie să întoarcă o coloană anume aşa că poate fi aleasă o constantă. Din punctul de vedere al performanței este mai rapidă selectarea unei constante.



Având coloana EMPLOYEE_ID în clauza SELECT a subinterrogării ar trebui parcursă tabela pentru această coloană. Prin înlocuirea ei cu constanta X, sau oricare alta, performanțele se îmbunătățesc, acesta fiind un mod eficient de folosire a operatorului IN.

Operatorul IN poate fi folosit ca modalitate alternativă pentru operatorul EXISTS, aşa cum demonstrează exemplul de mai jos:



```
SELECT employee_id, last_name, job_id, department_id
FROM employees
      WHERE employee_id IN (SELECT manager_id
                           FROM employees WHERE manager_id IS NOT NULL);
```

10.16 Folosirea operatorului NOT EXISTS



Exemplu : Afisați toate departamentele ce nu au angajați.

```
SELECT department_id, department_name  
      FROM departments d  
 WHERE NOT EXISTS (SELECT 'X'  
                      FROM employees  
                     WHERE department_id = d.department_id);
```

DEPARTMENT_ID	DEPARTMENT_NAME
190	Contracting

Se poate folosi o construcție de tip **NOT IN** drept alternativă pentru operatorul **NOT EXISTS**, așa cum arată exemplul de mai jos.



```
SELECT department_id, department_name  
      FROM departments  
 WHERE department_id NOT IN  
       (SELECT department_id FROM employees);
```

Dealtfel **NOT IN** evaluează FALSE dacă oricare membru al setului este o valoare de **NULL**. Cu atât mai mult interogarea nu va returna niciun rând chiar dacă sunt rânduri în tabelă care să satisfacă condiția din clauza **WHERE**.

10.17 Clauza WITH

Folosind clauza **WITH** puteți defini un bloc pentru interogare înainte de a îl folosi în aceasta. Clauza **WITH** (cunoscută formal ca **subquery_factoring_clause**) permite refolosirea același bloc într-o instrucție **SELECT** atunci când apare de mai multe ori într-o interogare complexă. În mod particular acest lucru este folositor atunci când există multe referințe la același bloc și sunt multe join-uri și calcule.

Folosind clauza **WITH**, puteți refolosi aceeași interogare atunci când costul evaluării blocului interogării este mare și apare de mai multe ori într-o interogare complexă. Folosind clauza **WITH** serverul Oracle regăsește rezultatele unui bloc și le stochează în spațiul de memorie temporară. Acest lucru duce la creșterea performanțelor, beneficiile fiind:

- interogarea devine ușor de citit;
- clauza este evaluată o singură dată chiar dacă apare de mai multe ori în interogare.



Exemplu: scrieți o interogare, folosind clauza **WITH**, care să afișeze numele departamentului și salariile totale pentru acele departamente în care salariul total este mai mare decât media salariilor pe departamente.

Problema ar putea necesita calcule intermediare și anume:

- Calculul salariului total pentru fiecare departament și stocarea rezultatelor folosind clauza WITH.
- Calculul mediei salariale pe departamente și stocarea rezultatului folosind clauza WITH.
- Compararea salariului total calculat la pasul unu cu media calculată în pasul doi. Dacă salariul total pentru un departament anume este mai mare decât media salariilor pe departamente, se afișează numele departamentului și salariul total pe acel departament.

```
WITH
dept_costs AS (
    SELECT department_name, SUM(salary) AS dept_total
    FROM employees, departments
    WHERE employees.department_id =
        departments.department_id
    GROUP BY department_name),
    avg_cost AS
    (SELECT SUM(dept_total)/COUNT(*) AS dept_avg
    FROM dept_costs)
SELECT * FROM dept_costs
WHERE dept_total >
(SELECT dept_avg FROM dept_avg)
ORDER BY department_name;

DEPARTMENT_NAME          DEPT_TOTAL
-----
Executive                  58000
Sales                      37100
```

Interrogarea de mai sus creează un bloc numit DEPT_COSTS și altul numit AVG_COST pe care le folosește apoi în interogarea principală. Intern, clauza WITH clausă este rezolvată fie printr-un view in-line sau printr-o tabelă temporară. Elementele de optimizare sunt cele care decid în funcție de costurile sau beneficiile de moment ale stocării temporare a datelor la folosirea acestei clauze.

Notă:



Subinterrogarea din clauza FROM a unei instrucțiuni SELECT se numește și in-line view. Clauza WITH se folosește numai la instrucțiuni SELECT.

Numele interogării este vizibil tuturor elementelor WITH ale blocurilor subinterrogărilor (inclusiv subinterrogărilor acestora) definite după acestea și blocului principal în sine (inclusiv subinterrogăriile).

Atunci când numele interogării este identic cu cel al unei tabele existente parserul caută de la interior spre exterior, numele blocului precedând numele tabelei.

Clauza WITH poate suporta mai mult de o interogare, fiecare interogare fiind separată de o virgulă.

10.18 Interogări ierarhice

Obiective:

- Interpretarea conceptului de interogare ierarhică;
- Crearea unui raport cu structură arborescentă ;
- Formatarea datelor ierarhice;
- Excluderea unor ramuri din structura arborescentă.

Folosind interogări ierarhice puteți regăsi date bazându-vă pe o relație ierarhică naturală, creată între rândurile dintr-un tabel.

O bază de date relațională nu stochează înregistrările în mod ierarhic. De altfel, acolo unde există o relație ierarhică între rândurile dintr-un singur tabel, puteți construi o ierarhie folosind procesul numit **parcursarea arborelui**.

O interogare ierarhică este o metodă de afișare a ramurilor unui arbore în ordine.

Imaginați-vă o familie în care cei mai vârstnici membri se află aproape de baza trunchiului arborelui iar cei mai tineri membri reprezintă ramurile arborelui. Ramurile pot avea alte ramuri și aşa mai departe.

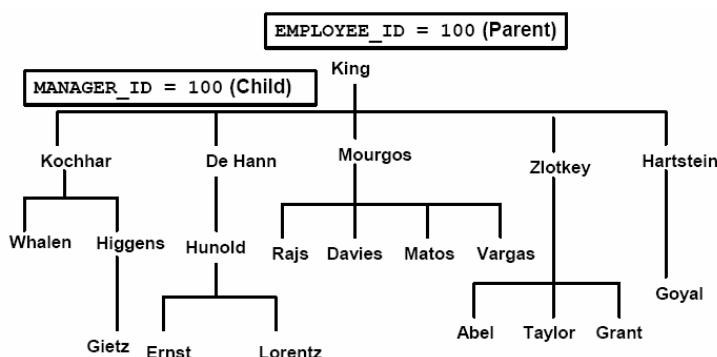
O interogare ierarhică este posibilă atunci când există o relație între rândurile tabelului. De exemplu, puteți observa în tabelul Employees că cei ce au job_ID-urile egale cu AD_VP, ST_MAN, SA_MAN și MK_MAN sunt direct subordonați președintelui companiei. Știm acest lucru deoarece coloana MANAGER_ID pentru aceste înregistrări are valoarea 100, care este de fapt valoarea coloanei employee_ID pentru președintele companiei (AD_PRES).

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice



Notă: interogările ierarhice pot fi folosite în diverse cazuri cum ar fi: genealogia umană (arborele genealogic al unei familii), managementul companiilor (managementul ierarhic), producție (asamblarea componentelor într-un produs finit), cercetarea evoluției (dezvoltarea speciilor) și cercetare științifică.

Structura naturală a arborelui



Tabelul EMPLOYEES are o structură arborescentă construită în funcție de subordonarea angajaților față de șefii lor. Ierarhia poate fi creată prin cercetarea valorilor echivalente din coloanele EMPLOYEE_ID și MANAGER_ID. Această relație poate fi folosită folosind un self-join pe acest tabel. Coloana MANAGER_ID conține numărul de marcă al șefului respectivului angajat. Relația părinte-copil a unei structuri ierarhice vă permite să controlați:

- Direcția în care se parcurge arborele;
- Punctul din cadrul ierarhiei de la care se pleacă.



NOTĂ: Figura de mai sus prezintă modul de organizare ierarhică al angajaților din tabelul EMPLOYEES.

Interogări ierarhice



```
SELECT [LEVEL], column, expr...
FROM table
[WHERE condition(s)]
[START WITH condition(s)]
[CONNECT BY PRIOR condition(s)];
WHERE condition:
expr comparison_operator expr
```

Interogările ierarhice pot fi identificate prin prezența clauzelor **CONNECT BY** și **START WITH**.

În sintaxă:

SELECT	este o instrucțiune SELECT standard.
LEVEL	pentru fiecare rând returnat de o interogare ierarhică, pseudocoloana LEVEL întoarce 1 pentru un rând rădăcină, 2 pentru un copil al unei rădăcini, etc.
FROM table	specifică tabelul sau vederea ce conține coloanele. Acestea pot fi selectate doar dintr-un singur tabel.
WHERE	restricționează rândurile date de interogare fără a afecta alte rânduri ale ierarhiei.
Condition	este compararea cu o expresie.
START WITH	specifică rândurile rădăcină ale arborelui, adică punctul de plecare. Această clauză este obligatorie pentru o interogare ierarhică adevărată.
CONNECT BY	specifică coloanele care conțin elementele ce definesc PRIOR rows relația dintre părinte și copil. Această clauză este obligatorie pentru o interogare ierarhică.

Instrucțiunea SELECT nu poate conține un join sau o interogare bazată pe un view care conține un join.

10.19 Parcurgerea arborelui – punctul de start

- Se specifică condițiile ce trebuie să îndeplinească;
- Se poate folosi în conjuncție cu orice altă condiție validă.



START WITH column1 = value

De exemplu, pentru tabela EMPLOYEES, să se parcurearborele începând cu angajatul al căruia nume este Kochhar.

...START WITH last_name = 'Kochhar'

Clauza **START WITH** precizează rândurile ce vor fi folosite ca bază a arborelui. Această clauză poate fi folosită în conjuncție cu orice altă clauză validă.

Exemple:



Folosind tabela EMPLOYEES, începeți cu King care este președintele companiei.

... START WITH manager_id IS NULL

Folosind tabela EMPLOYEES, începeți cu angajatul Kochhar. Condiția pusă în clauza START WITH poate conține o subinterrogare.

**... START WITH employee_id = (SELECT employee_id
FROM employees WHERE last_name = 'Kochhar')**

Dacă se omite clauza START WITH, parcurgerea arborelui are la bază toate rândurile din tabel, acestea fiind luate în considerare drept puncte de pornire.

Dacă este folosită o clauză WHERE, parcurgerea arborelui începe de la toate rândurile care satisfac condiția WHERE, ceea ce nu reflectă o ierarhie reală.



NOTĂ: Clauzele CONNECT BY PRIOR și START WITH nu sunt ANSI SQL standard.

CONNECT BY PRIOR column1 = column2

Parcurgerea tablei EMPLOYEES de sus în jos



**CONNECT BY PRIOR column1 = column2
... CONNECT BY PRIOR employee_id = manager_id**

Direcția de parcurgere arborelui

De sus în jos Column1 = Cheia părinte Column2 = Cheia copil

De jos în sus Column1 = Cheia copil Column2 = Cheia părinte

Direcția de parcurgere a arborelui, fie că este de la părinte la copil sau de la copil la părinte, este determinată de modul de scriere al coloanelor în clauza CONNECT BY PRIOR. Operatorul PRIOR se referă la rândul părinte. Pentru a determina un copil pentru un rând părinte, serverul Oracle evaluează expresia PRIOR pentru rândul părinte și cealaltă expresie pentru fiecare rând din tabel. Rândurile pentru care condiția este îndeplinită sunt copii pentru părinte. Serverul Oracle selectează întotdeauna copii prin evaluarea condiției CONNECT BY din punctul de vedere al îndeplinirii cerinței legate de rândul părinte.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice



Exemplu: definiți o relație ierarhică parcurgând tabelul EMPLOYEES de sus în jos, astfel încât valoarea pentru EMPLOYEE_ID din rândul părinte să fie egală cu cea pentru MANAGER_ID din rândul copil.

... **CONNECT BY PRIOR employee_id = manager_id**

Pentru a parurge tabela de jos în sus condiția devine :

... **CONNECT BY PRIOR manager_id = employee_id**

Operatorul PRIOR nu necesită cod, în mod special, imediat după CONNECT BY. De exemplu expresia din exemplul de mai sus poate fi scrisă și sub forma de mai jos, rezultatul fiind același.

... **CONNECT BY employee_id = PRIOR manager_id**



Atenție : Clauza CONNECT BY nu poate conține o subinterrogare.

Parcurgerea arborelui de jos în sus



```
SELECT employee_id, last_name, job_id, manager_id
FROM employees
START WITH employee_id = 101
CONNECT BY PRIOR manager_id = employee_id;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
101	Kochhar	AD_VP	100
100	King	AD_PRES	

Exemplul afișează lista şefilor începând cu angajatul al căruia ID este 101.

În următorul exemplu, valorile pentru EMPLOYEE_ID sunt evaluate pentru rândul părinte și MANAGER_ID iar valorile salariailor sunt evaluate pentru rândurile copil. Operatorul PRIOR este aplicat doar asupra valorilor din coloana EMPLOYEE_ID.

... **CONNECT BY PRIOR employee_id = manager_id AND salary > 15000;**

Pentru a fi calificat drept rând copil, acel rând trebuie să aibă valoarea din coloana MANAGER_ID egală cu valoarea coloanei EMPLOYEE_ID din rândul

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

părinte și trebuie să aibă salariul mai mare de \$15,000.

Parcurgerea arborelui de sus în jos

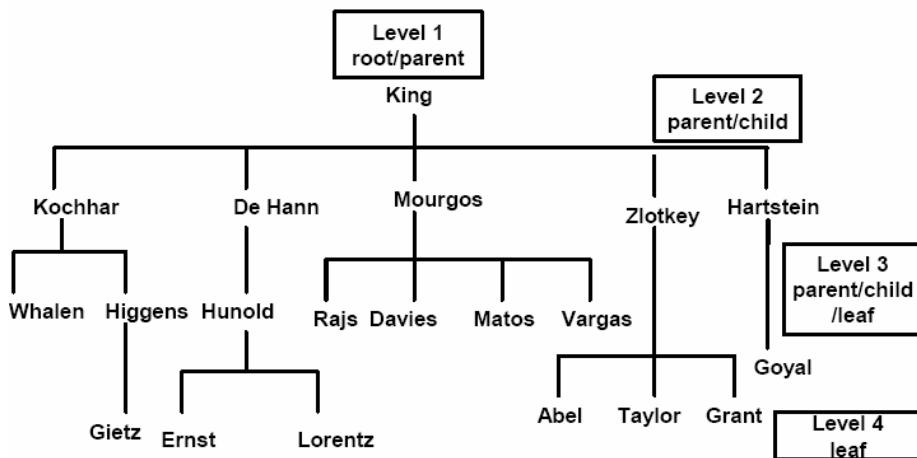


```
SELECT last_name||' reports to '||
PRIOR last_name "Walk Top Down"
FROM employees START WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id;
```

Walk Top Down
King reports to
Hartstein reports to King
Fay reports to Hartstein
Kochhar reports to King
Whalen reports to Kochhar
Mourgos reports to Zlotkey
Vargas reports to Mourgos
Zlotkey reports to King
Abel reports to Zlotkey
Taylor reports to Zlotkey
Grant reports to Zlotkey

20 rows selected.

Folosirea pseudocoloanei LEVEL pentru clasificarea rândurilor



Puteți afișa în mod explicit rangul sau nivelul unui rând în ierarhie folosind pseudocoloana LEVEL.

Aceasta va face ca raportul dvs. să fie mult mai explicit. Locurile unde

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

una sau mai multe ramuri se despart dintr-o ramură mare sunt numite noduri iar locul unde o ramură ia sfârșit se numește frunză sau frunza nodului. Diagrama de mai sus prezintă nodurile unui arbore întors precum și valorile pentru fiecare nivel. De exemplu, angajatul Higgens este și părinte și copil pe când angajatul Davies este doar un copil și o frunză.



Notă: un nod rădăcină este nodul de cel mai înalt nivel dintr-un arbore inversat. Un nod copil este un nod care nu este rădăcină. Un nod părinte este un nod care are copii. Un nod frunză este orice nod care nu are copii. Numărul nivelelor ce este returnat de o interogare ierarhică poate fi limitat de memoria disponibilă. În exemplu, King este rădăcina sau nodul părinte (LEVEL = 1). Kochhar, De Hann, Mourgos, Zlotkey, Hartstein, Higgens și Hunold sunt copii și deasemeni părinți (LEVEL = 2). Whalen, Rajs, Davies, Matos, Vargas, Gietz, Ernst, Lorentz, Abel, Taylor, Grant și Goyal sunt copii și frunze (LEVEL = 3 și LEVEL = 4).

Valorile pseudocoloanei LEVEL sunt:

- Nod rădăcină;
- Copil al unui nod rădăcină;
- Copil al unui copil , etc.

Formatarea rapoartelor ierarhice folosind LEVEL și LPAD



Creați un raport care să afișeze nivelele de management ale companiei începând de la nivelul cel mai înalt și identificați fiecare nivel de subordonare.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')
AS org_chart FROM employees
START WITH last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

Se asignează nodurilor arborelui câte un număr de nivel folosind funcția LPAD împreună cu pseudocoloana LEVEL pentru a afișa un raport ierarhic sub forma unui arbore identat.

În exemplul de mai sus :

- LPAD(*char1,n [,char2]*) întoarce *char1*, aliniat la stânga pe lungime de *n* caractere folosind secvența de caractere din *char2*. Argumentul *n* reprezintă lungimea totală a valorii returnate aşa cum este afişată pe ecran.
- LPAD(*last_name*,*LENGTH(last_name)+(LEVEL*2)-2,'_'*) definește formatul de afișare.

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

- *char1* este LAST_NAME , *n* este lungimea totală a valorii returnate, adică lungimea lui LAST_NAME +(LEVEL*2)-2 iar *char2* este '_.

Cu alte cuvinte i se spune motorului SQL să scrie la stânga valorii LAST_NAME atâtea caractere '' câte sunt date de valoarea lui LENGTH(last_name)+(LEVEL*2)-2. Pentru King, LEVEL = 1 deci $(2 * 1) - 2 = 2 - 2 = 0$, aşadar King nu va avea niciun caracter '_' și este afișat în coloana 1. Pentru Kochhar, LEVEL = 2 deci $(2 * 2) - 2 = 4 - 2 = 2$ aşadar Kochhar va avea 2 caractere '_' și este afișat identat. Celelalte rânduri sunt afișate după același principiu.

ORG_CHART	
King	
Kochhar	
Whalen	
Higgins	
Gietz	
De Haan	
Hunold	
Ernst	
Lorentz	
Mourgos	
Rajs	
Davies	
Matos	
Vargas	
Zlotkey	
Abel	
Taylor	
Grant	
Hartstein	
Fay	

20 rows selected.

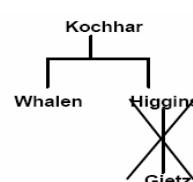
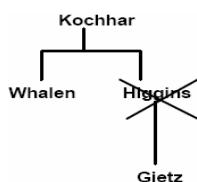
Tăierea ramurilor

Folosiți clauza WHERE pentru a elimina un nod.

WHERE last_name != 'Higgins'

Folosiți clauza CONNECT BY pentru a elimina o ramură.

*CONNECT BY PRIOR
employee_id = manager_id
AND last_name != Higgins'*



Puteți folosi clauzele WHERE și CONNECT BY pentru a tăia arborele; acesta înseamnă că puteți controla care noduri și rânduri vor fi afișate. Predicatul pe care îl folosiți acționează ca o condiție Booleană.

Exemplu: Pornind de la rădăcină, parcurgeți arborele de sus în jos și eliminați angajatul Higgins din rezultat procesând însă rândurile copil.

 **SELECT department_id, employee_id, last_name, job_id, salary
FROM employees WHERE last_name != 'Higgins' START WITH
manager_id IS NULL CONNECT BY PRIOR employee_id = manager_id;**

Pornind de la rădăcină, parcurgeți arborele de sus în jos și eliminați angajatul Higgins din rezultat și toate rândurile copil.

**SELECT department_id, employee_id, last_name, job_id, salary
FROM employees START WITH manager_id IS NULL CONNECT BY PRIOR
employee_id = manager_id AND last_name != 'Higgins';**



10.20 Exerciții

- Scriți o interogare care să afișeze numele angajatului și data angajării pentru toți angajații din același departament ca Zlotkey, excludând-ul pe Zlotkey.

LAST_NAME	HIRE_DATE
Abel	11-MAY-96
Taylor	24-MAR-98

- Scriți o interogare pentru a afișa numărul angajatului și numele său pentru toți angajații care câștigă mai mult decât salariul mediu. Sortați rezultatele în ordinea descrescătoare a salariului.

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000
149	Zlotkey	10500
174	Abel	11000
205	Higgins	12000
201	Hartstein	13000
101	Kochhar	17000
102	De Haan	17000
100	King	24000

8 rows selected.

- Scriți o interogare care să afișeze numărul și numele angajatului pentru toți cei care lucrează într-un departament care deține cel puțin un angajat al

EMPLOYEE_ID	LAST_NAME
124	Mourgos
141	Rajs
142	Davies
143	Matos
144	Vargas
103	Hunold
104	Ernst
107	Lorentz

cărui nume conține litera ‘ u ’. Salvați instrucțiunea într-un fișier denumit pq3.sql.

- Afișați numele angajatului, numărul departamentului și funcția pe care lucrează acesta pentru toți angajații al căror ID de departament este 1700.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

LAST_NAME	DEPARTMENT_ID	JOB_ID
Whalen	10	AD_ASST
King	90	AD_PRES
Kochhar	90	AD_VP
De Haan	90	AD_VP
Higgins	110	AC_MGR
Gietz	110	AC_ACCOUNT

6 rows selected.

5. Afișați numele și salariul tuturor angajaților subordonați lui King.

LAST_NAME	SALARY
Kochhar	17000
De Haan	17000
Mourgos	5800
Zlotkey	10500
Hartstein	13000

6. Afișați numărul departamentului, numele și funcția tuturor angajaților din departamentul numit „Executive”.

DEPARTMENT_ID	LAST_NAME	JOB_ID
90	King	AD_PRES
90	Kochhar	AD_VP
90	De Haan	AD_VP

7. Modificați pq3.sql pentru a afișa numărul, numele și salariul tuturor angajaților care câștigă mai mult decât salariul mediu și totodată lucrează într-un departament care deține cel puțin un angajat ce conține în numele sau litera ‘u’. Salvați fișierul ca pq7.sql. Execuați interogarea din nou.

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000

8. Scrieți o interogare care să afișeze numele, numărul departamentului și salariul oricărui angajat al căruia număr de departament și salariu să se potrivească cu numărul departamentului și salariul oricărui angajat care câștigă comision.

LAST_NAME	DEPARTMENT_ID	SALARY
Taylor	80	8600
Zlotkey	80	10500
Abel	80	11000

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

9. Afipați numele, numele departamentului și salariul oricărui angajat al căruia salaria și comision se potrivesc cu salariul și comisionul oricărui angajat care lucrează în location ID=1700.

LAST_NAME	DEPARTMENT_NAME	SALARY
Whalen	Administration	4400
Gietz	Accounting	8300
Higgins	Accounting	12000
Kochhar	Executive	17000
De Haan	Executive	17000
King	Executive	24000

6 rows selected.

10. Scrieți o interogare care să afișeze numele, data angajării și salariul pentru toți angajații care au același salaria și comision ca al lui Kochhar, exclusiv Kochhar.

LAST_NAME	HIRE_DATE	SALARY
De Haan	13-JAN-93	17000

11. Scrieți o interogare care să afișeze angajații care câștigă un salaria mai mare ca al oricărui angajat cu JOB_ID = 'SA_MAN'. Afipați salariile ordonat de la cel mai mare la cel mai mic.

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Kochhar	AD_VP	17000
De Haan	AD_VP	17000
Hartstein	MK_MAN	13000
Higgins	AC_MGR	12000
Abel	SA REP	11000

6 rows selected.

12. Afipați coloanele employee_ID, last_name și department_ID pentru acei angajații care locuiesc în orașe a căror nume începe cu T.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
201	Hartstein	20
202	Fay	20

13. Scrieți o interogare care să afișeze toți angajații care câștigă mai mult decât salaria mediu din departamentul lor. Afipați coloanele last_name, salary, department_ID și media salarială pentru departament. Ordonați datele după salaria mediu. Folosiți alias-uri pentru coloanele returnate de interogare, după exemplul de mai jos.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

ENAME	SALARY	DEPTNO	DEPT_AVG
Mourgos	5800	50	3500
Hunold	9000	60	6400
Harstein	13000	20	9500
Abel	11000	80	10033.3333
Zlotkey	10500	80	10033.3333
Higgins	12000	110	10150
King	24000	90	19333.3333

7 rows selected.

14. Afișați toți angajații ce nu sunt supervisors.
 a. Folosiți operatorul NOT EXISTS.

LAST_NAME
Ernst
Lorentz
Rajs
Davies
Matos
Vargas
Abel
Taylor
Grant
Whalen
Fay
Gietz

12 rows selected.

- b. Se poate folosi operatorul NOT IN ? Cum, sau de ce nu ?

15. Scrieți o interogare care să afișeze numele angajaților care câștigă mai puțin decât media salarială din departamentul lor.

LAST_NAME
Kochhar
De Haan
Ernst
Lorentz
Davies
Matos
Vargas
Taylor
Fay
Gietz

10 rows selected.

16. Scrieți o interogare care să afișeze numele angajaților care au unul sau mai

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

mulți colegi de departament care să aibă salarii mai mari și date de angajare mai târzii.

LAST_NAME
Rajs
Davies
Matos
Vargas
Taylor

17. Scrieți o interogare care să afișeze coloanele employee _ID, last_name și department_name pentru toți angajații. Folosiți în instrucțiunea SELECT o subinterrogare scalară pentru a găsi numele departamentului.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT
205	Higgins	Accounting
206	Gietz	Accounting
200	Whalen	Administration
100	King	Executive
101	Kochhar	Executive
102	De Haan	Executive
103	Hunold	IT
104	Ernest	IT
107	Lorentz	IT
201	Hartstein	Marketing
202	Fay	Marketing
149	Zlotkey	Sales
176	Taylor	Sales
174	Abel	Sales
124	Mourgos	Shipping
141	Rajs	Shipping
142	Davies	Shipping
143	Matos	Shipping
144	Vargas	Shipping
178	Grant	

20 rows selected.

18. Scrieți o interogare care să afișeze numele de departament pentru acele departamente al căror cost total cu salariile este peste 1/8 din costul total cu salariile al întregii companii. Folosiți clauza WITH. Numiți interogarea SUMMARY.

DEPARTMENT_NAME DEPT_TOTAL

Executive 58000
Sales 37100

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

19. Priviți rezultatele de mai jos. Sunt acestea rezultate în urma procesării unei subinterrogări ierarhice ? Explicați răspunsul dat.

RANK	EMPLOYEE_ID	DEPARTMENT_ID	MANAGER_ID
1	100	90	
2	101	90	100
3	200	10	101
3	205	110	101
4	206	110	205
2	102	90	100
3	103	60	102
4	104	60	103

3	174	60	149
3	176	80	149
3	178		149
2	201	20	100
3	202	20	201

EMPLOYEE_ID	LAST_NAME	MANAGER_ID	SALARY	DEPARTMENT_ID
144	Vargas	124	2500	50
143	Matos	124	2600	50
142	Davies	124	3100	50
141	Rajs	124	3500	50
107	Lorentz	103	4200	60
200	Whalen	101	4400	10
124	Mourgos	100	5800	50
104	Ernst	103	6000	60
202	Fay	201	6000	20

201	Hartstein	100	13000	90
101	Kochhar	100	17000	90
102	De Haan	100	17000	90
100	King		24000	90

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
200	Whalen	10	Administration
201	Hartstein	20	Marketing
202	Fay	20	Marketing
124	Mourgos	50	Shipping
141	Rajs	50	Shipping

100	King	90	Executive
101	Kochhar	90	Executive
102	De Haan	90	Executive
205	Higgins	110	Accounting
206	Gietz	110	Accounting

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

20. Afipați structura organizațională a departamentului lui Mourgos specificând numele de familie, salariul și id-ul departamentului.

LAST_NAME
Hunold
De Haan
King

21. Creați un raport care să afișeze ierarhia managerilor pentru angajatul Lorentz. Afipați mai întâi pe șeful său direct.

LAST_NAME	SALARY	DEPARTMENT_ID
Mourgos	5800	50
Rajs	3500	50
Davies	3100	50
Matos	2600	50
Vargas	2500	50

22. Creați un raport identat care să afișeze ierarhia managerială începând de la angajatul cu numele Kochhar. Afipați coloanele last_name, manager_ID și department_ID. Dați aliasuri coloanelor ca în exemplu.

NAME	MGR	DEPTNO
Kochhar	100	90
Whalen	101	10
Higgins	101	110
Gietz	205	110

23. Afipați structura organizatorică a companiei afișând ierarhia managerială. Începeți cu persoanele de la nivelul superior, excludeți toți angajații ce au job_ID egal cu IT_PROG și excludeți-l pe De Haan și toți subordonații lui.

LAST_NAME	EMPLOYEE_ID	MANAGER_ID
King	100	
Hartstein	201	100
Fay	202	201
Kochhar	101	100
Whalen	200	101
Higgins	205	101
Gietz	206	205
Mourgos	124	100
Rajs	141	124
Davies	142	124
Matos	143	124
Vargas	144	124
Zlotkey	149	100
Abel	174	149
Taylor	176	149
Grant	178	149

16 rows selected.

Capitolul 11 Instrucțiuni pentru Manipularea Datelor

Obiective:

- Descrierea fiecărei comenzi DML(**Data Manipulation Language**);
- Inserarea de înregistrări într-un tabel (**INSERT**);
- Actualizarea înregistrărilor dintr-un tabel (**UPDATE**);
- Ștergerea înregistrărilor dintr-un tabel (**DELETE**);
- Alăturarea rândurilor unei tabele (**MERGE**);
- Controlul tranzacțiilor (**COMMIT**, **SAVEPOINT** și **ROLLBACK**).

O comandă DML este executată atunci când:

- Se adăuga noi înregistrări în tabelă;
- Se modifica înregistrările existente într-o tabelă;
- Se sterg înregistrări existente dintr-o tabelă.

O tranzacție constă dintr-o colecție de comenzi DML care formează o unitate logică de lucru.

Limbajul de manipulare a datelor (DML) este partea de bază a SQL. Când se dorește adăugarea, modificarea sau ștergerea datelor dintr-o baza de date, se execută o comandă DML. O colecție de comenzi DML care formează o unitate logică de lucru se numește **tranzacție**.

Exemplu: Considerați o bază de date din domeniul bancar. Atunci când un client al băncii dorește să transfere bani dintr-un depozit într-un cont curent, tranzacția ar putea consta în 3 operații separate: scăderea sumei din depozit, creșterea sumei din contul curent, înregistrarea tranzacției în jurnalul de tranzacții. Serverul Oracle trebuie să garanteze că toate cele 3 comenzi SQL sunt executate în aşa fel încât să mențină echilibrul necesar între conturi. Atunci când, din anumite cauze, una dintre comenziile tranzacției de mai sus nu se execută, atunci celelalte comenzi ale tranzacției trebuie să fie anulate.

Adăugarea unei noi înregistrări într-un tabel.

Imaginea de mai jos ilustrează adăugarea unui nou departament în tabelul DEPARTMENTS.

Facultatea de Automatică și Calculatoare Iași Baze de date – lucrări practice

New row

DEPARTMENTS			
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

Insert a new row
into the
DEPARTMENTS table.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

11.1 Introducerea datelor-comanda INSERT

Folosind comanda **INSERT** se pot adăuga noi înregistrări într-un tabel.



INSERT INTO table [(column [, column...])] VALUES (value [, value...]);

În descrierea sintaxei:

- tbl* este numele tabelului din baza de date.
- coloana* este numele coloanei din tabelul respectiv.
- valoare* este valoarea corespunzătoare coloanei.



Notă: Folosind această sintaxă pentru comanda INSERT se adaugă numai câte un rând odată la un tabel.

La inserarea unei noi înregistrări conținând valori pentru fiecare coloană, valorile se dau în ordinea prestabilită a coloanelor din tabel, precizarea coloanelor fiind optională.



**INSERT INTO departments(department_id, department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);**

DESCRIBE departments

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)



Pentru claritate, precizați lista coloanelor în sintaxa INSERT.
Încadrați între ghilimele simple doar sirurile de caractere și datele calendaristice nu și valorile numerice.

Inserarea de înregistrări cu valori de NULL

Metoda implicită: Omiterea coloanei din listă.



```
INSERT INTO departments (department_id,  
department_name )  
VALUES (30, 'Purchasing');
```

Metoda explicită:

- Specificarea cuvântului cheie **NULL** în lista de la clauza VALUES.
- Specificați sirul vid (") în lista VALUES (numai pentru siruri de caractere și pentru valori de tip dată calendaristică).



```
INSERT INTO departments  
VALUES (100, 'Finance', NULL, NULL);
```



Asigurați-vă că pentru coloana vizată este permisă valoarea **NULL**.

Serverul Oracle aplică automat toate constrângerile de integritate, de domeniu și tip pentru date. Orice coloană care nu este specificată explicit în listă, va primi o valoare nulă în noua înregistrare.

Erorile uzuale ce apar la introducerea datelor sunt :

- Lipsa unei valori obligatoriu a fi introdusă pentru o coloană ce nu admite valori de NULL ;
- Valori multiple care încalcă constrângerea de unicitate;
- Valori care duc la încălcarea constrângerii de tip cheie străină ;
- Valori care duc la încălcarea constrângerii de tip CHECK ;
- Tip de data eronat;
- Valoare mai mare decât maxima admisă pentru acea coloană.

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

Inserarea unor valori speciale

Funcția **SYSDATE** furnizează data și timpul curent.



```
INSERT INTO EMPLOYEES (EMPLOYEE_id,first_name,  
last_name,email,phone_number,hire_date,job_id,  
salary,commission_pct,manager_id,department_id)  
VALUES (113,'Louis','Popp','LPOPP','515.124.4567',  
SYSDATE, 'AC_ACCOUNT', 6900,NULL, 205, 100);
```

Exemplul de mai sus înregistrează informația pentru angajatul Popp în tabela EMPLOYEES. Pentru a introduce data și ora curentă în câmpul HIREDATE este folosită funcția **SYSDATE**.

Se poate de asemenei folosi funcția **USER** la introducerea de valori într-un tabel. Funcția **USER** furnizează numele utilizatorului curent.

Verificarea datelor adăugate în tabel:

```
SELECT EMPLOYEE_id, last_name, job_id, hire_date,  
commission_pct FROM EMPLOYEES WHERE EMPLOYEE_id = 113;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
113	Popp	AC_ACCOUNT	12-MAR-01	

Inserarea unor valori specifice de tip dată calendaristică



Exemplu: Adăugarea unui nou angajat

```
INSERT INTO EMPLOYEES VALUES  
(114,'Den','Raphealy','DRAPHEAL','515.127.4561', TO_DATE ('FEB 3,  
1999', 'MON DD, YYYY'), 'AC_ACCOUNT', 11000, NULL, 100, 30);
```

Formatul DD-MON-YY este de obicei folosit pentru a insera o valoare de tip dată calendaristică. Cu acest format, secolul este implicit cel curent. Deoarece data conține de asemenea informații despre timp, ora implicită este 00:00:00.

Dacă o dată calendaristică necesită specificarea altui secol sau oră, trebuie folosită funcția **TO_DATE**.

În exemplu se înregistrează informația despre angajatul Raphealy în tabela EMPLOYEES. Câmpul HIREDATE primește valoarea February 3, 1999.

Inserarea de valori folosind variabile de substituție

Se poate scrie o comandă INSERT care să permită utilizatorului să

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

adauge valori în mod interactiv folosind variabilele de substituție SQL*Plus.

De exemplu, se introduc informațiile pentru un nou departament în tabelul DEPARTMENTS. Numărul departamentului, numele și locația sunt cerute interactiv utilizatorului.

Pentru valori de tip dată calendaristică sau sir de caractere, ampersandul (&) și numele variabilei sunt încadrate de ghilimele simple (apostrof).



```
INSERT INTO departments  
(department_id, department_name, location_id)  
VALUES (&department_id, '&department_name',&location);
```

Crearea unui script

Un script este un fișier cu extensia sql în care sunt memorate instrucțiuni.

Comenzi, care pot să conțină și variabile de substituție, pot fi salvate într-un fișier. Acestea poate fi executat și la fiecare execuție sunt cerute valori noi pentru variabile. La execuția comenzi SQL*Plus ACCEPT, valorile date variabilelor pot fi diferite, deci se poate folosi același script pentru a introduce valori diverse în tabelă.



ACCEPT expr PROMPT 'textul specificat'

Comanda ACCEPT memorează valoarea introdusă de utilizator în variabila expr iar PROMPT afișează 'textul specificat'.



```
ACCEPT department_id PROMPT 'Introduceți numărul  
departamentului:'  
ACCEPT department_name PROMPT 'Introduceți numele  
departamentului:'
```

ACCEPT location_id PROMPT 'Introduceți orașul:'

```
INSERT INTO departments  
(department_id, department_name, location_id)  
VALUES (&department_id, '&department_name',&location);
```

Parametrul de substituție SQL*Plus nu trebuie precedat de & când este referit într-o comandă **ACCEPT**. Pentru a continua o comandă SQL*PLUS pe linia următoare se folosește caracterul -.

Copierea înregistrărilor dintr-un alt tabel folosind comanda INSERT cu o subinterrogare

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

Comanda INSERT poate fi folosită pentru a adăugă înregistrări într-un tabel, valorile pentru câmpuri fiind extrase dintr-un tabel existent. Pentru aceasta se folosește în locul clauzei VALUES o subinterrogare.



```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT EMPLOYEE_id, last_name, salary, commission_pct
FROM EMPLOYEES WHERE job_id LIKE "%REP%";
```

4 rows created.



INSERT INTO tabel [coloana (, coloana)] Subinterrogare;

unde:

tabel este numele tabelului din baza de date

coloana este numele coloanei din tabelul în care se face inserarea.

subinterrogare este subinterrogarea care returnează înregistrările în tabel.



Numărul și tipul câmpurilor (coloanelor) din lista specificată în comanda INSERT trebuie să corespundă numărului și tipului valorilor din subinterrogare.

Folosirea subinterrogarilor în comanda INSERT



```
INSERT INTO
(SELECT employee_id, last_name, email, hire_date, job_id, salary,
department_id FROM employees WHERE department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
TO_DATE('07-JUN-99', 'DD-MON-RR'), 'ST_CLERK', 5000, 50);
```

1 row created.

În cadrul comenzi INSERT se poate folosi o subinterrogare în locul numelui tabelului. Instrucțiunea SELECT a subinterrogării trebuie să aibă același număr de coloane ca și cel din clauza VALUES. Pentru ca INSERT să fie executată satisfăcător trebuie să țineți cont de toate regulile care se aplică asupra coloanelor din tabelul de bază. De exemplu, nu puteți introduce valori duplicate sau nu este permisă lipsa unei valori pentru o coloană care este obligatoriu NOT NULL.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

11.2 Modificarea datelor - comanda UPDATE

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMIS.
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-88	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	18-NOV-98	ST_MAN	5800	60	

Update rows in the EMPLOYEES table.



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMIS.
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	30	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	30	
107	Diana	Lorentz	DLORENTZ	07-FEB-88	IT_PROG	4200	30	
124	Kevin	Mourgos	KMOURGOS	18-NOV-98	ST_MAN	5800	30	

Figura de mai sus arată modificarea numărului departamentului pentru cei din departamentul 60 în 30.

Înregistrările existente pot fi modificate folosind comanda **UPDATE**.



UPDATE table
SET column = value [, column = value,...]
[WHERE condition];

În sintaxa de mai sus:

tabel este numele tabelului;

coloana este numele coloanei din tabelul în care se face modificarea;

Valoare este noua valoare sau o subinterrogare ce produce noua valoare pentru coloana ce se modifică;

Condiție identifică înregistrările care trebuie modificate și este alcătuită din expresii, nume de coloane, constante, subinterrogări și operatori de comparare.

Folosind comanda **UPDATE** se pot modifica simultan valorile pentru unul sau mai multe rânduri din tabelă.

Puteți avea confirmarea executării operației de modificare prin interogarea

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

tabelului, afișând rândurile modificate.



NOTĂ: În general, folosiți cheia primara pentru a identifica o singură înregistrare. Folosirea altor coloane poate determina modificarea mai multor înregistrări.

De exemplu, identificarea unei singure înregistrări în tabelul EMPLOYEES prin nume poate fi periculoasă, deoarece pot exista mai mulți angajați cu același nume.

Comanda UPDATE modifică **anumite** înregistrări dacă este specificată clauza **WHERE**. Exemplul următor transferă angajatul cu numărul 113 în departamentul 70.



```
UPDATE employees  
SET department_id = 70  
WHERE employee_id = 113;  
1 row updated.
```

Dacă omiteți clauza WHERE, sunt modificate toate înregistrările dintr-un tabel.



```
UPDATE copy_emp  
SET department_id = 110;  
22 rows updated.
```

Actualizarea înregistrărilor folosind subinterrogări după mai multe câmpuri

În clauza SET a unei comenzi **UPDATE** pot fi implementate subinterrogări după mai multe câmpuri.



```
UPDATE table  
SET column = (SELECT column  
FROM table WHERE condition)  
[, column =  
    (SELECT column  
     FROM table  
     WHERE condition)]  
[WHERE condition];
```



Modifică pentru angajatul cu numărul 114 funcția și salariul astfel încât ele să coincidă cu cele ale angajatului numărul 205.

```
UPDATE employees SET
job_id = (SELECT job_id FROM employees WHERE employee_id = 205),
salary = (SELECT salary FROM employees WHERE employee_id = 205)
WHERE employee_id = 114;
1 row updated.
```

Actualizarea înregistrărilor folosind valori dintr-un alt tabel

Exemplul de mai jos arată cum se pot folosi subinterrogări în comenzi **UPDATE** pentru a actualiza înregistrările dintr-un tabel pe baza valorilor din alt tabel.



```
UPDATE copy_emp
SET department_id = (SELECT department_id FROM employees
WHERE employee_id = 100)
WHERE job_id = (SELECT job_id FROM employees
WHERE employee_id = 200);
1 row updated.
```

Încălcarea constrângerii de integritate



```
UPDATE employees
SET department_id = 55 WHERE department_id = 110;
```

Dacă încercați să atribuiți unui câmp o valoare care este legată de o constrângere de integritate, va rezulta o eroare. În exemplul de mai sus, departamentul cu numărul 55 nu există în tabelul părinte, DEPARTMENTS, și astfel veți obține eroarea *parent key violation ORA-02291* aşa cum se arată mai jos.

```
UPDATE employees
*
```

ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated-parent key not found



NOTĂ: Constrângerile de integritate asigură faptul că datele aderă la un set predefinit de reguli. Un capitol următor va dezvolta acest subiect mai pe larg.

11.3 Ștergerea datelor - comanda *DELETE*

Folosind comanda ***DELETE*** se pot șterge înregistrări dintr-un tabel.



***DELETE [FROM] table
[WHERE condition];***

În sintaxa:

tbl este numele tabelei.

Condiție identifică înregistrările care trebuie șterse și este alcătuită din expresii, nume de coloane, constante, subinterrogări și operatori de comparare.

Se pot șterge doar anumite înregistrări prin specificarea clauzei ***WHERE*** în comanda ***DELETE***. Exemplul de mai jos arată cum se procedează pentru a șterge departamentul '**Finance**' din tabelul DEPARTMENTS. Vă puteți asigura că ștergerea s-a făcut corect, încercând afișarea rândurilor șterse printr-un SELECT.



***DELETE FROM departments
WHERE department_name = 'Finance';***
1 row deleted.

```
SELECT *  
FROM departments WHERE department_name = 'Finance';  
no rows selected.
```



DELETE FROM copy_emp;

Dacă se omite clauza WHERE se vor șterge toate rândurile din tabela.

Ștergerea înregistrărilor folosind valori dintr-un alt tabel

Folosiți subinterrogări în comanda ***DELETE*** pentru a șterge înregistrările dintr-un tabel pe baza valorilor din alt tabel.



***DELETE FROM employees
WHERE department_id = (SELECT department_id
FROM departments WHERE department_name LIKE
'%Public%');***

1 row deleted.

Puteți folosi subinterrogări pentru a șterge înregistrări dintr-un tabel, folosind informațiile din alt tabel. Exemplul de mai sus șterge toți angajații care sunt în departamentul ce conține în numele sau șirul 'Public'. Subinterrogarea caută în tabela DEPARTMENTS numărul de departament pentru 'Public', apoi furnizează numărul de departament interrogării principale, care șterge înregistrări din EMPLOYEES pe baza acestuia.

Încălcarea constrângerii de integritate

Dacă încercați ștergerea unei înregistrări care conține un câmp cu o valoare legată de o constrângere de integritate, veți obține o eroare.

În exemplul de mai jos se încearcă ștergerea departamentului cu numărul 60 din tabelul DEPARTMENTS. Ștergerea provoacă o eroare deoarece numărul de departament este folosit ca și cheie externă în tabelul EMPLOYEES. Dacă înregistrarea părinte pe care încercați să o ștergeți are înregistrări copil, atunci veți primi un mesaj de eroare: child record found violation ORA-02292.



```
DELETE FROM departments
WHERE department_id = 60;
DELETE FROM departments
*
```

ERROR at line 1:
ORA-02292: integrity constraint (HR.EMP_DEPT_FK)
violated-child record found



Nu puteți șterge o înregistrare care conține o cheie primară folosită ca cheie externă în alt tabel.

Folosirea sintagmei WITH CHECK OPTION în instrucțiuni de tip DML

În exemplul de mai jos se folosește o subinterrogare pentru a identifica tabelul și coloanele pentru instrucțiunea DML.

Sintagma **WITH CHECK OPTION** împiedică schimbarea rândurilor care nu sunt în subinterrogare.



```
INSERT INTO (SELECT employee_id, last_name,
email,hire_date, job_id, salary FROM employees
WHERE department_id = 50 WITH CHECK OPTION)
VALUES (99998, 'Smith', 'JSMITH',TO_DATE('07-JUN-99',
'DD- MON-RR'),'ST_CLERK', 5000);
```

```
INSERT INTO
*
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Specificați **WITH CHECK OPTION** pentru a indica faptul că, dacă se folosește o subinterrogare în locul numelui tabelului într-o instrucțiune de tip INSERT, UPDATE sau DELETE nu sunt permise nici un fel de schimbări asupra rândurilor care nu sunt în subinterrogare.

În exemplu este folosită sintagma **WITH CHECK OPTION**. Subinterrogarea identifică rândurile care sunt în departamentul 50, dar department_ID nu este în lista de la SELECT deci nu este prevăzută o valoare pentru department_id în lista clauzei VALUES. Înserarea acestui rând, care are pentru department_ID valoarea null, ar duce la înserarea unei valori care nu este în subinterrogare.

Folosirea opțiunii DEFAULT explicit

Folosind opțiune DEFAULT explicit puteți să folosiți cuvântul cheie DEFAULT drept valoare pentru o coloană acolo unde aceasta este necesară. Introducerea acestei opțiuni a fost făcută în scopul alinierii la cerințele standardului SQL: 1999. Ea permite utilizatorului să controleze locul și timpul la care opțiunea DEFAULT va fi aplicată datelor.

Opțiunea DEFAULT explicit poate fi folosită în instrucțiunile INSERT și UPDATE pentru a identifica o valoare implicită pentru coloană. Dacă nu există o valoare implicită se va folosi valoarea NULL.



DEFAULT cu INSERT

```
INSERT INTO departments
(department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```



DEFAULT cu UPDATE

```
UPDATE departments
SET manager_id = DEFAULT WHERE department_id = 10;
```

În primul exemplu, instrucțiunea INSERT folosește o valoare implicită pentru coloana MANAGER_ID. Dacă nu este definită nicio valoare implicită pentru această coloană se va inseră o valoare de NULL.

Cel de-al doilea exemplu folosește instrucțiunea UPDATE pentru a atribui coloanei MANAGER_ID valoarea implicită pentru cei din departamentul 10.

Dacă nu este definită o valoare implicită, valoarea pentru coloana MANAGER_ID va fi NULL.



Valoarea implicită pentru o coloană se specifică în momentul creării unei tabele.

11.4 Instrucțiunea MERGE

- Prevede posibilitatea de a introduce sau modifica datele din tabel în mod condiționat;
- Execută un UPDATE dacă rândul există și un INSERT dacă este un rând nou;
- Previne scrierea mai multor instrucțiuni UPDATE ;
- Este ușor de folosit și duce la creșterea performantelor;
- Este folositoare în aplicații ce gestionează volum mare de date.

SQL a fost extins în vederea introducerii instrucțiunii MERGE. Folosind aceasta instrucțiune se pot modifica sau insera rânduri într-o tabelă în mod condiționat ceea ce previne scrierea mai multor instrucțiuni UPDATE. Decizia de a modifica sau introduce rânduri în tabela țintă se bazează pe clauza condițională ON.

Deoarece instrucțiunea MERGE combină instrucțiunile INSERT și UPDATE, trebuie să aveți drepturi pentru modificarea sau adăugarea de rânduri în tabela țintă și dreptul de a selecta datele din tabela sursă.

Instrucțiunea MERGE este deterministă. Nu puteți să faceți UPDATE de mai multe ori asupra acelaiași rând din tabela folosind aceeași instrucțiune MERGE.

O abordare alternativă ar fi folosirea buclelor PL/SQL și mai multe instrucțiuni DML. Instrucțiunea MERGE este de altfel ușor de folosit și mult mai simplu de scris fiind o singură instrucțiune de tip SQL.

Instrucțiunea MERGE este potrivită pentru unele aplicații de tip warehousing. De exemplu, într-o asemenea aplicație s-ar putea să fiți nevoiți să lucrați cu date ce provin din mai multe surse, unele dintre ele putând fi duplicate. Folosind instrucțiunea MERGE puteți adăuga sau modifica rândurile în mod condiționat.



```
MERGE INTO table_name AS table_alias  
USING (table|view|sub_query) AS alias  
ON (join condition)  
WHEN MATCHED THEN  
    UPDATE SET  
    col1 = col_val1,  
    col2 = col2_val  
WHEN NOT MATCHED THEN  
    INSERT (column_list)  
    VALUES (column_values);
```

În sintaxa :

INTO specifică tabela către pe care se face UPDATE sau INSERT ;
USING identifică sursa datelor ce vor fi modificate sau inserate; poate fi o tabelă, view sau subinterrogare ;
ON clauză condițională pe baza căreia MERGE execută fie modificarea fie inserarea datelor ;
WHEN MATCHED dă instrucțiuni serverului asupra felului în care să Sau răspundă la rezultatul condiției de join.
WHEN NOT MATCHED



```
MERGE INTO copy_emp AS c  
USING employees e  
ON (c.employee_id = e.employee_id)  
WHEN MATCHED THEN  
    UPDATE SET  
    c.first_name = e.first_name,  
    c.last_name = e.last_name,  
    ...  
    c.department_id = e.department_id  
WHEN NOT MATCHED THEN  
    INSERT VALUES(e.employee_id, e.first_name, e.last_name,  
    e.email, e.phone_number, e.hire_date, e.job_id,  
    e.salary, e.commission_pct, e.manager_id,  
    e.department_id);
```

În exemplul de mai sus este evaluată condiția c.employee_id = e.employee_id. Deoarece tabela COPY_EMP nu are înregistrări, condiția returnează false deci se execută clauza WHEN NOT MATCHED iar instrucțiunea MERGE inserează rândurile din tabela EMPLOYEES în tabela COPY_EMP.

Dacă tabela COPY_EMP ar conține rânduri care ar avea în coloana employee_ID valori egale cu cele din tabela EMPLOYEES, aceste rânduri din tabela COPY_EMP ar fi modificate pentru a conține valori identice cu cele din tabela EMPLOYEES.

11.5 Tranzacții

Tranzacțiile constau dintr-un grup format din următoarele comenzi:

- Comenzi DML care duc la o schimbare consistentă a datelor;
- O comanda DDL;
- O comanda DCL.

Serverul Oracle asigură consistența datelor pe baza tranzacțiilor. Tranzacțiile oferă mai multă flexibilitate și control la modificări aduse datelor și asigură consistența datelor în eventuala cădere a sistemului sau în cazul unei erori a procesului utilizator.

Tranzacțiile constau din comenzi DML care realizează o schimbare consistentă asupra datelor. De exemplu, un transfer de fonduri între două conturi ar trebui să includă modificarea debitului unui cont și creditului celuilalt cont cu aceeași sumă de bani. Ambele acțiuni ar trebui fie să reușească împreună, fie să eșueze împreună. Operația de creditare nu ar trebui să poată fi efectuată fără ca cea de debitare să fie executată cu succes.

Tipuri de tranzacții:

Tip	Descriere
Limbaj de manipulare a datelor (DML)	Consta din orice număr de comenzi DML pe care serverul Oracle le tratează ca o singură entitate sau unitate logică de lucru.
Limbaj de definire a datelor (DDL)	Conștă dintr-o singură comandă DDL
Limbaj de control al datelor (DCL)	Conștă dintr-o singură comandă DCL

Tranzacții în baze de date:

- Începe odată cu execuția primei comenzi SQL executabile;
- Se termină la apariția unuia dintre următoarele evenimente:
 - COMMIT sau ROLLBACK;
 - Execuția unei comenzi DDL sau DCL (automatic commit);
 - Deconectarea utilizatorului;
 - Căderea sistemului.

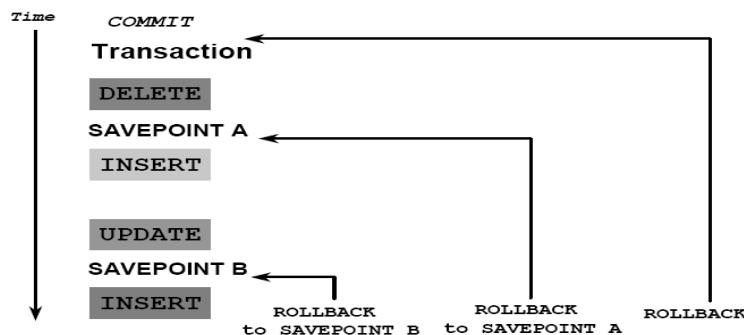
După ce se încheie o tranzacție, următoarea tranzacție va începe automat la prima comandă SQL executabilă întâlnită. Rezultatele unei comenzi

DDL sau DCL sunt salvate automat (automat *commit*) motiv pentru care tranzacția se încheie în mod implicit.

Avantajele comenziilor COMMIT și ROLLBACK:

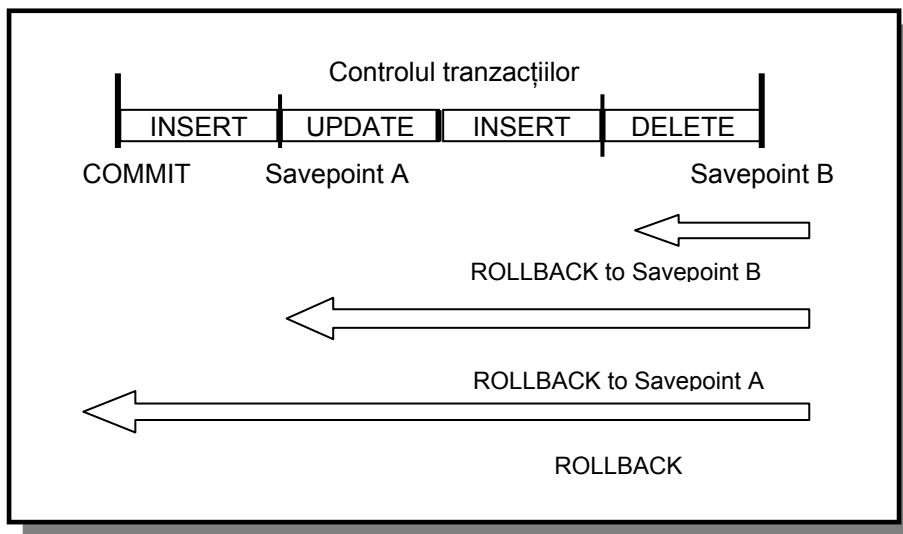
- Asigură consistența datelor;
- Oferă posibilitatea revederii schimbărilor făcute înainte de a le salva;
- Grupează operații relaționate logic.

Controlul tranzacțiilor



Comenzi pentru controlul explicit al tranzacțiilor

Puteți controla logica tranzacțiilor folosind comenziile;
COMMIT, SAVEPOINT și ROLLBACK.



Comanda	Descriere
COMMIT	Încheie actuala tranzacție făcând ca toate modificările să devină permanente (salvează modificările)
SAVEPOINT nume	Marchează un punct de întoarcere (savepoint) în cadrul tranzacției curente
ROLLBACK [TO SAVEPOINT name]	Comanda ROLLBACK începe tranzacția curentă, pierzându-se toate modificările temporare (pending changes) asupra datelor. ROLLBACK TO SAVEPOINT name șterge savepoint-ul și toate schimbările de după el (temporare).



Notă: SAVEPOINT nu este ANSI standard SQL.



UPDATE... SAVEPOINT update_done;

Savepoint created.

INSERT... ROLLBACK TO update_done;

Rollback complete.

Procesarea implicită a tranzacțiilor

Un commit (salvarea modificărilor) automat are loc în următoarele circumstanțe:

- Este dată o comandă DDL;
- Este dată o comandă DCL;
- O părăsire normală a mediului SQL*Plus, fără a da explicit o comandă COMMIT sau ROLLBACK.

Un rollback automat are loc în condițiile unei terminări anormale a sesiunii SQL*Plus sau în cazul unei 'căderi' de sistem.

Stare	Circumstanțe
Commit automat	Comanda DDL sau DCL lăsată normală din SQL*Plus, fără o comandă COMMIT sau ROLLBACK explicită
Rollback automat	Terminare anormală a SQL*Plus sau cădere sistem



Notă: În SQL*Plus mai este disponibilă și o a treia comandă. Comanda SQL*Plus AUTOCOMMIT poate fi setată ON sau OFF. Dacă este setată pe ON, fiecare comandă DML individuală duce la salvarea modificărilor, imediat ce este executată. Nu se mai poate reveni la situația dinainte (un rollback nu mai este posibil). Dacă este setată pe OFF, COMMIT poate fi dată explicit. De asemenei, COMMIT este executată odată cu o comandă DDL sau la ieșirea din SQL*Plus.

Căderile sistemului: Când o tranzacție este întreruptă de o cădere a sistemului, întreaga tranzacție este automat pierdută (este 'rolled back'). Aceasta împiedică ca eroarea să determine schimbări nedorite asupra datelor și reface starea bazelor din momentul ultimului COMMIT (explicit sau implicit). Astfel, SQL*Plus păstrează integritatea tabelelor (bazelor de date).

Salvarea modificărilor

Fiecare modificare efectuată în timpul tranzacției este temporară până la apariția unui 'commit' (până la 'salvarea' tranzacției).

Starea datelor înainte de un COMMIT sau ROLLBACK

Operațiile de manipulare a datelor afectează inițial buffer-ul bazei de date; de aceea, starea inițială a datelor poate fi refăcută.

Utilizatorul curent poate urmări schimbările făcute prin interogarea tabelelor.

Alți utilizatori nu pot vedea modificările făcute de utilizatorul curent. Serverul Oracle instituie o **consistență la citire** pentru a se asigura că fiecare utilizator vede datele aşa cum existau ele în momentul ultimei salvări.

Înregistrările afectate sunt protejate (locked); alți utilizatori nu pot face modificări asupra lor.

Starea datelor după COMMIT

- Modificările asupra datelor din baza de date devin permanente.
- Starea anterioară a datelor nu mai poate fi refăcută.
- Toți utilizatorii pot vedea rezultatele.
- Înregistrările protejate (locked) sunt deblocate pentru modificare și pot fi schimbate de alți utilizatori.
- Toate savepoint-urile sunt șterse.



Exemplu : Faceți modificările:

DELETE FROM employees WHERE employee_id = 99999;

1 row deleted.

INSERT INTO departments VALUES (290, 'Corporate Tax', NULL, 1700);

1 row inserted.

Salvați modificările

COMMIT;

Commit complete.

Exemplul de mai sus șterge din tabelul EMPLOYEES rândul ce are employee_id egal cu 99999 și apoi introduce în tabela DEPARTMENTS un rând nou. Prin folosirea comenții COMMIT modificările devin permanente.

Starea datelor după ROLLBACK

Pentru a anula modificările temporare făcute folosiți comanda ROLLBACK.

- Modificările aduse datelor sunt pierdute.
- Starea anterioară a datelor este refăcută.
- Protecția asupra înregistrărilor implicate este ridicată.



Exemplu:

Încercând să ștergeți o înregistrare din tabelul TEST, puteți șterge accidental întreg tabelul. Puteți corecta greșeala, iar apoi să dați comenziile corecte și să salvați modificările.

DELETE FROM test;

25,000 rows deleted.

ROLLBACK;

Rollback complete.

DELETE FROM test WHERE id = 10

1 row deleted.

SELECT * FROM test WHERE id = 100;

No rows selected.

COMMIT;

Commit complete.

Anularea modificărilor pana la un savepoint

Puteți crea un marcat în cadrul tranzacției curente folosind comanda SAVEPOINT. Astfel, tranzacția poate fi împărțită în secțiuni mai mici. Puteți apoi anula modificările temporare până la acel marcat folosind comanda ROLLBACK TO SAVEPOINT.

Dacă creați un al doilea savepoint cu același nume ca unul anterior,

savepoint-ul anterior este șters.

Rollback la nivel de comandă

Se poate anula o parte din tranzacție printr-un rollback implicit dacă este detectată o eroare la execuția unei comenzi. Dacă o singură comandă DML eșuează în timpul execuției unei tranzacții, efectul ei este anulat printr-un rollback la nivel de comandă, dar schimbările făcute de comenziile DML anterioare în tranzacție nu vor fi anulate. Ele pot fi salvate (committed) sau anulate (rolled back) în mod explicit de către utilizator.

Oracle execută o comandă COMMIT implicită înainte și după orice comandă DDL.

Deci, chiar dacă comanda DDL nu se execută cu succes, nu puteți anula comenziile anterioare pentru că serverul a executat un commit (a salvat modificările).

Este bine ca tranzacțiile să fie explicit finalizate prin executarea unei comenzi COMMIT sau ROLLBACK.

11.6 Consistența la citire

- Consistența la citire garantează o vedere consistentă datelor în fiecare moment.
- Schimbările făcute de un utilizator nu intră în conflict cu schimbările realizate de un altul.
- Asigură că, pentru aceleasi date:
 - Cei care citesc datele nu trebuie să îi aștepte pe cei care le modifică;
 - Cei care modifică datele să nu trebuie să îi aștepte pe cei care le citesc.

Utilizatorii bazei de date accesează tabelele din baza în două moduri:

- Operații de citire (comanda SELECT);
- Operații de scriere (comenzi INSERT, UPDATE, DELETE).

Consistența la citire este necesară pentru ca:

- Cei care citesc/modifică datele să aibă o vedere consistentă a datelor;
- Cei care citesc datele să nu vadă datele care sunt în curs de modificare;
- Cei care modifică datele să aibă siguranță că schimbările în baza de date se fac în mod consistent;
- Schimbările făcute de un utilizator să nu intre în conflict sau să afecteze schimbările făcute de un altul.

Scopul consistenței la citire este să asigure că fiecare utilizator vede datele în starea în care erau la ultima salvare, înainte să înceapă o operație DML.



Consistența la citire este implementată în mod automat. Este păstrată o copie parțială a bazei de date în segmente rollback.

Când se realizează o operație de adăugare, actualizare sau ștergere asupra bazei de date, serverul Oracle scrie o copie a datelor dinainte de modificare într-un *segment rollback*.

Toți utilizatorii, cu excepția celui care a inițiat modificarea, văd baza de date în starea de dinaintea începerii modificării; ei vad datele din segmentul rollback.

Înainte ca schimbările să fie salvate în baza de date, numai utilizatorul care modifică datele vede baza de date modificată, toți ceilalți văzând datele din segmentul rollback. Aceasta garantează că utilizatorii citesc consistent datele care nu suferă schimbări chiar în acel moment.

Când o comandă DML este salvată, schimbarea făcută în baza de date devine vizibilă oricui execută o comandă SELECT. Spațiul ocupat de 'vechile' date din segmentul rollback este eliberat pentru a fi reutilizat.

Dacă tranzacția este anulată, schimbările sunt la rândul lor anulate.

Versiunea originală mai veche a datelor din segmentul rollback este scrisă înapoi în tabel. Toți utilizatorii văd baza de date aşa cum era înainte de a începe tranzacția.

Protecția la scriere (locking)

Protecția Oracle:

- Previne interacțiunile destructive între tranzacții concurente;
- Nu necesită acțiuni din partea utilizatorului;
- Folosește în mod automat cel mai mic nivel de restricționare;
- Este valabilă pe durata unei tranzacții.

Există două modalități de protecție : Exclusivă și Partajată.

Protecțiile (locking) sunt mecanisme care previn interacțiunea destructivă între tranzacții ce accesează aceeași resursă: fie un obiect utilizator (de exemplu tabele sau înregistrări), fie obiecte sistem care nu sunt vizibile utilizatorilor (de exemplu structuri de date partajate și înregistrări "data dictionary").

Cum protejează Oracle datele

Protejarea unei baze de date Oracle este automatizată în întregime și nu necesită acțiuni din partea utilizatorului. Implicit, protejarea are loc pentru toate comenzi SQL cu excepția lui SELECT. Mecanismul implicit de protecție în Oracle folosește în mod automat cel mai mic nivel aplicabil de restricționare, furnizând astfel cel mai mare grad de concurență existent, precum și integritatea maxima a datelor. Oracle permite și protejarea manuală a datelor de către utilizator.

Oracle folosește două moduri de protecție într-o bază de date multiutilizator.

Mod de protecție	Descriere
exclusivă (exclusiv lock)	Împiedică partajarea unei resurse. Prima tranzacție care blochează resursa în mod exclusiv este singura tranzacție care poate modifica resursa până când protecția exclusivă este anulată.
partajată (share lock)	Permite partajarea resursei. Mai mulți utilizatori care citesc datele le pot folosi în comun prin crearea unor protecții partajate ce împiedică accesul concurrent pentru scriere (care necesită o protecție exclusivă). Mai multe tranzacții pot obține protecții partajate pentru aceeași resursă.

11.7 Correlated UPDATE

În cazul instrucțiunii UPDATE se poate folosi o subinterrogare corelată pentru a modifica rândurile dintr-un tabel pe baza valorilor dintr-un alt tabel.



```
UPDATE table1 alias1  
SET column = (SELECT expression  
FROM table2 alias2  
WHERE alias1.column = alias2.column);
```



```
ALTER TABLE employees  
ADD(department_name VARCHAR2(14));
```

```
UPDATE employees e  
SET department_name =  
(SELECT department_name  
FROM departments d  
WHERE e.department_id = d.department_id);
```

Facultatea de Automatică și Calculatoare Iași

Baze de date – lucrări practice

Exemplul de mai sus adăugă o coloană la tabelul EMPLOYEES pentru numele departamentului și apoi introduce valori în această coloană folosind un UPDATE corelat.



Folosiți o comandă UPDATE corelat pentru a modifica rândurile din tabela EMPLOYEES pe baza valorilor din tabela REWARDS.

```
UPDATE EMPLOYEES
SET salary = (SELECT EMPLOYEES.salary +
rewards.pay_raise
FROM rewards
WHERE EMPLOYEE_id =
EMPLOYEES.EMPLOYEE_id
AND payraise_date =
(SELECT MAX(payraise_date)
FROM rewards
WHERE
EMPLOYEE_id = EMPLOYEES.EMPLOYEE_id))
WHERE EMPLOYEES.EMPLOYEE_id
IN (SELECT EMPLOYEE_id FROM rewards);
```

Acest exemplu folosește tabela REWARDS care are coloanele EMPLOYEE_ID, PAY_RAISE și PAYRAISE_DATE. De fiecare dată când un angajat primește o creștere de salarit se înregistrează în această tabelă un rând ce conține valoarea pentru EMPLOYEE_ID, valoarea creșterii salariale și data la care a fost executată. Tabela REWARDS poate conține mai multe rânduri care să aibă aceeași valoare pentru câmpul EMPLOYEE_ID. Coloana PAYRAISE_DATE are menirea de a identifica cea mai recentă creștere salarială primită de un angajat. În exemplu, coloana SALARY din tabela EMPLOYEES este actualizată astfel încât să reflecte ultima creștere salarială primită de un angajat. Acest lucru se realizează prin adăugarea la salariul actual a măririi de salarit corespunzătoare din tabela REWARDS.

11.8 Correlated DELETE



```
DELETE FROM table1 alias1
WHERE column operator
(SELECT expression FROM table2 alias2
WHERE alias1.column = alias2.column);
```

În cazul unei instrucțiuni DELETE se poate folosi o subinterrogare corelată pentru a șterge acele rânduri care există și într-o altă tabelă.



De exemplu, dacă decideți ca în tabela JOB_HISTORY să păstrați doar ultimele patru înregistrări pentru un angajat, atunci când un angajat se transferă pe o altă funcție, veți șterge cel mai vechi rând din tabelă privitor la angajatul în discuție prin căutarea valorii MIN(START_DATE). Exemplul ilustrează rezolvarea acestei probleme folosind un DELETE corelat.

```
DELETE FROM job_history JH
WHERE EMPLOYEE_id =
(SELECT EMPLOYEE_id FROM EMPLOYEES E
WHERE JH.EMPLOYEE_id = E.EMPLOYEE_id
AND START_DATE = (SELECT MIN(start_date)
FROM job_history JH
WHERE JH.EMPLOYEE_id = E.EMPLOYEE_id)
AND 5 > (SELECT COUNT(*)
FROM job_history JH
WHERE JH.EMPLOYEE_id = E.EMPLOYEE_id
GROUP BY EMPLOYEE_ID
HAVING COUNT(*) >= 4));
```



Folosiți o subinterrogare corelată pentru a șterge doar acele rânduri din tabela EMPLOYEES care există și în tabela EMPLOYEES_HISTORY.

```
DELETE FROM EMPLOYEES E
WHERE EMPLOYEE_id = (SELECT EMPLOYEE_id
FROM EMPLOYEES_history WHERE EMPLOYEE_id =
E.EMPLOYEE_id);
```

În acest exemplu se folosesc două tabele:

- Tabela EMPLOYEES care conține detalii despre toți angajații;
- Tabela EMPLOYEES_HISTORY care conține detalii despre funcțiile anterior deținute de angajați, aşa că ar fi o greșală dacă aceleași înregistrări despre un angajat ar exista în ambele tabele.

Instrucțiunile Multitable INSERT

Instrucțiunea INSERT...SELECT poate fi folosită pentru a introduce rânduri în mai multe tabele ca parte a unei singure instrucțiuni DML.

Instrucțiunea **Multitable INSERT** poate fi folosită în sistemele de tip data warehousing pentru a transfera date de la una sau mai multe surse operaționale către un set de tabele țintă.

Instrucțiunea **Multitable INSERT** oferă o îmbunătățire semnificativă a performanțelor din următoarele puncte de vedere:

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

- o singură instrucțiune DML versus multiple instrucțiuni INSERT.. SELECT ;
- o singură instrucțiune DML versus o procedură care să execute mai multe instrucțiuni INSERT folosind sintaxa IF...THEN.

Într-o instrucțiune multitable INSERT inserați în una sau mai multe tabele rândurile calculate derive din rândurile returnate în urma evaluării datelor furnizate de o subinterrogare. Instrucțiunea multitable INSERT poate juca un rol deosebit de folositor într-un scenariu pentru data warehouse. Presupunem că trebuie să încărcați în mod regulat datele în baza pentru a facilita analiza afacerii. Pentru a face asta trebuie să extrageți datele din unul sau mai multe sisteme operaționale și să le copiați în depozitul de date. Procesul de extragere a datelor din surse și aducerea lor în depozit se numește de ETL (Extraction, Transformation, and Loading).

În timpul extragerii, datele necesare trebuie identificate și extrase din mai multe surse diferite cum ar fi sisteme de baze de date și aplicații. După extragere, datele trebuie transportate fizic către sistemul țintă sau un sistem intermediar pentru a fi procesate. În funcție de modul ales pentru transport pot fi executate unele transformări în timpul acestui proces. De exemplu, o instrucțiune SQL care accesează direct o țintă la distanța printr-un gateway poate concatena două coloane ca parte a instrucțiunii SELECT.

Odată ce datele sunt încărcate în baza Oracle9i transformarea lor poate fi făcută folosind operații SQL. Instrucțiunea multitable INSERT este una dintre tehniciile implementate în Oracle9i pentru transformarea datelor.

Instrucțiunea Multitable INSERTS oferă beneficiile date de instrucțiunea INSERT... SELECT atunci când sunt folosite mai multe tabele drept țintă. Înainte de Oracle9i trebuia să folosiți mai multe instrucțiuni independente de tipul INSERT... SELECT pentru a procesa aceeași data sursă de mai multe ori, timpul aferent și munca necesara fiind indubabil mai mare.

Fiecare înregistrare dintr-un sir de intrare, cum ar fi o tabelă non-relațională poate fi acum convertită în mai multe înregistrări pentru mai multe tabele relaționale.

Oracle 9i introduce următoarele tipuri de instrucțiuni multitable INSERT :

- **Unconditional INSERT** (INSERT necondiționat);
- **Conditional ALL INSERT** (Insearea tutore datelor în mod condiționat);
- **Conditional FIRST INSERT** (Primul INSERT condiționat);
- **Pivoting INSERT** (INSERT condus).

Trebuie să folosiți diferite clauze pentru a indica tipul de INSERT ce va fi executat.



***INSERT [ALL] [conditional_insert_clause]
[insert_into_clause values_clause] (subquery)
conditional_insert_clause[ALL] [FIRST]
[WHEN condition THEN] [insert_into_clause values_clause]
[ELSE] [insert_into_clause values_clause]***

Sintaxa de mai sus redă formatul generic pentru instrucțiunea multitable INSERT.

Unconditional INSERT clauza ALL

Specificați clauza ALL urmată de mai multe clauze insert_into pentru a executa un insert necondițional în mai multe tabele. Serverul Oracle execută fiecare clauza INSERT odată pentru fiecare rând returnat de subinterrogare.

Conditional INSERT: conditional_insert_clause

Specificați clauza pentru conditional_insert_clause în vederea executării unui insert condiționat. Serverul Oracle filtrează fiecare clauză insert_into_clause căutând corespondența cu condiția de la clauza WHEN care determină execuția ei. O singură instrucțiune multitable insert poate conține până la 127 clauze WHEN.

Conditional INSERT: ALL

Dacă specificați ALL, serverul Oracle evaluează fiecare clauza WHEN cu privire la rezultatul evaluării altor clauze WHEN. Pentru fiecare clauza WHEN a cărei condiție este evaluată ca fiind adevărată serverul Oracle execută lista corespunzătoare din instrucțiunea INSERT.

Conditional FIRST: INSERT

Dacă specificați FIRST, serverul Oracle evaluează fiecare clauza WHEN în ordinea în care sunt scrise în instrucțiune. Dacă primul WHEN este evaluat ca fiind adevărat, serverul Oracle execută clauza INTO corespunzătoare, în caz contrar trecând peste următoarea clauză WHEN pentru acel rând.

Conditional INSERT: clauza ELSE

Pentru un anumit rând, dacă nici o clauza WHEN nu este evaluată ca fiind adevărată :

- dacă specificați clauza ELSE, serverul Oracle execută lista de la clauza INTO asociată clauzei ELSE.
- dacă nu specificați clauza ELSE, serverul Oracle nu execută nici o acțiune pentru acel rând.



Restricții pentru instrucțiunea Multitable INSERT

- Instrucțiunea multitable insert poate fi executată numai asupra tabelelor nu și asupra view-urilor sau tabelelor la distanță (remote table);
- Nu puteți specifica o expresie formată dintr-o colecție de tabele când folosiți instrucțiunea multitable insert;
- La folosirea instrucțiunii multitable insert nu pot fi specificate mai mult de 999 tabele întă, luând în calcul toate clauzele insert_into.



Exemplu: Unconditional INSERT ALL

Selectați valorile pentru EMPLOYEE_ID, HIRE_DATE, SALARY și MANAGER_ID din tabela EMPLOYEES pentru acei angajați care au EMPLOYEE_ID > 200. Inserați valorile în tabela SAL_HISTORY și în tabela MGR_HISTORY folosind un INSERT multitable.

```
INSERT ALL
INTO sal_history VALUES (EMPLOYEESID, HIREDATE, SAL)
INTO mgr_history VALUES (EMPLOYEESID, MGR, SAL)
SELECT EMPLOYEE_id EMPLOYEESID, hire_date HIREDATE,
salary SAL, manager_id MGR
FROM EMPLOYEES
WHERE EMPLOYEE_id > 200;
```

În exemplu se inserează rânduri atât în tabela SAL_HISTORY cât și în tabela MGR_HISTORY tables.

Instrucțiunea SELECT extrage datele din tabela EMPLOYEES, coloanele EMPLOYEE_ID, hire_date, salary și manager_ID pentru acei angajați a căror EMPLOYEE_ID este mai mare de 200. Aceste valori sunt inserate în tabela SAL_HISTORY și în tabela MGR_HISTORY.

Acest INSERT este necondiționat deoarece ulterior nu se aplica restricții asupra rândurilor extrase de instrucțiunea SELECT. Toate rândurile regăsite de instrucțiunea SELECT statement sunt inserate în cele două tabele, respectiv SAL_HISTORY și MGR_HISTORY. Clauza VALUES de la instrucțiunea INSERT specifică coloanele ce vor fi inserate în tabele. Fiecare rând returnat de instrucțiunea SELECT duce la două inserări, una în tabela SAL_HISTORY și alta în tabela MGR_HISTORY. În total au fost inserate 8 rânduri în cele două tabele.



Exemplu-Conditional INSERT ALL

Selectați valorile coloanelor EMPLOYEE_ID, HIRE_DATE, SALARY și MANAGER_ID din tabela EMPLOYEES pentru acei angajați a căror id este mai mare 200. Dacă SALARY este mai mare de 10,000, inserați aceste valori în tabela SAL_HISTORY folosind o instrucție multitable INSERT condițională. Dacă valoarea pentru MANAGER_ID este mai mare de 200 atunci inserați aceste valori în tabela MGR_HISTORY folosind o instrucție multitable INSERT condițională.

```
INSERT ALL
WHEN SAL > 10000 THEN
    INTO sal_history VALUES(EMPID,HIREDATE,SAL)
WHEN MGR > 200 THEN
    INTO mgr_history VALUES(EMPID,MGR,SAL)
    SELECT employee_id EMPID,hire_date HIREDATE ,
    salary SAL , manager_id MGR
FROM employees
WHERE employee_id > 200;
4 rows created.
```

Exemplul este similar cu cel anterior deoarece se introduc date în două tabele și anume SAL_HISTORY și MGR_HISTORY. Instrucțunea SELECT extrage din tabela EMPLOYEES datele despre angajații (ID, hire date, salary, and manager_ID) a căror employee_ID este mai mare decât 200. Valorile coloanelor employee_ID, hire_date și salary sunt inserate în tabela SAL_HISTORY iar cele din coloanele employee_ID, manager_ID și salary sunt inserate în tabela MGR_HISTORY.

Această instrucție INSERT este referită ca „conditional ALL INSERT” deoarece sunt aplicate restricții după extragerea rândurilor de către instrucție SELECT. Numai acele rânduri care au valoarea din coloana SAL mai mare de 10000 sunt inserate în tabela SAL_HISTORY și similar, numai acele rânduri care au valoarea din coloana MGR mai mare de 200 sunt inserate în tabela MGR_HISTORY.

Față de exemplul precedent, de această dată sunt inserate numai patru rânduri în cele două tabele SAL_HISTORY și MGR_HISTORY.

Conditional FIRST INSERT



```
INSERT FIRST
WHEN SAL > 25000 THEN
    INTO special_sal VALUES(DEPARTMENTSID, SAL)
WHEN HIREDATE like ('%00%') THEN
    INTO hiredate_history_00 VALUES(DEPARTMENTSID, HIREDATE)
WHEN HIREDATE like ('%99%') THEN
    INTO hiredate_history_99 VALUES(DEPARTMENTSID, HIREDATE)
ELSE
    INTO hiredate_history VALUES(DEPARTMENTSID, HIREDATE)
SELECT department_id DEPARTMENTSID, SUM(salary) SAL,
       MAX(hire_date) HIREDATE
  FROM EMPLOYEES
 GROUP BY department_id;
```

În exemplu instrucțiunea SELECT extrage valorile pentru coloanele department_ID, salariul total și valoarea maximă pentru hire_date pentru fiecare departament din tabela EMPLOYEES.

Aceasta instrucțiune INSERT este referită ca „conditional FIRST INSERT”, deoarece atunci când salariul total este mai mare de 25,000 apare o excepție. Expresia condițională WHEN ALL > 25000 este prima evaluată. Dacă salariul total pentru un departament este mai mare de 25,000 atunci articolul este inserat în tabela SPECIAL_SAL în funcție de valoarea coloanei hire_date. Dacă prima clauză WHEN este evaluată ca fiind True, serverul Oracle execută clauza corespondentă INTO și trece peste următoarele clauze WHEN pentru acest rând.

Pentru rândurile care nu satisfac condiția WHEN SAL > 25000, restul condițiilor sunt evaluate ca fiind o instrucțiune de tip „conditional INSERT” și rândurile date de SELECT sunt inserate în tabela HIREDATE_HISTORY_00 sau HIREDATE_HISTORY_99 sau HIREDATE_HISTORY în funcție de valoarea din coloana HIREDATE.

Se execută 8 inserări în tabelele SPECIAL_SAL,
HIREDATE_HISTORY_00, HIREDATE_HISTORY_99 și
HIREDATE_HISTORY.



Pivoting INSERT

- Presupunem că primiți un set de date despre vânzări dintr-o bază de date nerelațională, SALES_SOURCE_DATA în următorul format : EMPLOYEE_ID, WEEK_ID, SALES_MON, SALES_TUE, SALES_WED,

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

SALES_THUR, SALES_FRI și doriți să păstrați aceste înregistrări în tabela SALES_INFO într-un format relațional cum ar fi : EMPLOYEE_ID, WEEK, SALES.

- Folosind o instrucție de tip „pivoting INSERT”, transformați setul de date primit în formatul de date relațional dorit.

Pivotarea este operația prin care creați o transformare astfel încât fiecare înregistrare din orice șir de intrare, cum ar fi o bază de date care nu este relațională, trebuie să fie convertit în mai multe înregistrări pentru o bază de date relațională. Pentru a soluționa acesta problemă trebuie să construiți o transformare astfel încât fiecare înregistrare din tabela originală din baza de date nerelațională, SALES_SOURCE_DATA, să fie convertită în cinci înregistrări pentru tabela SALES_INFO din baza de date relațională. Această operație este numită de obicei „pivoting”.

```
INSERT ALL
INTO sales_info VALUES (EMPLOYEE_id,week_id,sales_MON)
INTO sales_info VALUES (EMPLOYEE_id,week_id,sales_TUE)
INTO sales_info VALUES (EMPLOYEE_id,week_id,sales_WED)
INTO sales_info VALUES
(EMPLOYEE_id,week_id,sales_THUR)
INTO sales_info VALUES (EMPLOYEE_id,week_id, sales_FRI)
SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
sales_WED, sales_THUR,sales_FRI
FROM sales_source_data;
```

DESC SALES_SOURCE_DATA

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
WEEK_ID		NUMBER(2)
SALES_MON		NUMBER(8,2)
SALES_TUE		NUMBER(8,2)
SALES_WED		NUMBER(8,2)
SALES_THUR		NUMBER(8,2)
SALES_FRI		NUMBER(8,2)

SELECT * FROM SALES_SOURCE_DATA;

EMPLOYEE_ID	WEEK_ID	SALES_MON	SALES_TUE	SALES_WED	SALES_THUR	SALES_FRI
176	6	2000	3000	4000	5000	6000

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

DESC SALES_INFO

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
WEEK		NUMBER(2)
SALES		NUMBER(8,2)

SELECT * FROM sales_info;

EMPLOYEE_ID	WEEK	SALES
176	6	2000
176	6	3000
176	6	4000
176	6	5000
176	6	6000



11.9 Exerciții

1. Creați tabela MY_EMPLOYEE care are următoarele coloane : id, last_name, first_name, userid, salary. Inserați date în tabelul MY_EMPLOYEE.
2. Descrieți structura tabelului MY_EMPLOYEE pentru a identifica numele câmpurilor.
3. Adăugați prima înregistrare a tabelului din exemplul de mai jos. Nu folosiți lista coloanelor în clauza INSERT.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	795
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audry	aropebur	1550

4. Introduceți în tabelul MY_EMPLOYEE și al doilea rând de date din exemplul de mai sus. De data aceasta, specificați coloanele explicit în clauza INSERT.
5. Verificați dacă au fost adăugate în tabel.
6. Creați un script numit load_EMPLOYEES.sql pentru a insera înregistrări în mod interactiv în tabelul MY_EMPLOYEE. Cereți utilizatorului prenumele (FIRST_NAME), numele de familie (LAST_NAME) și salariul fiecărui angajat. Concatenați prima literă a prenumelui și primele 7 caractere ale numelui de familie pentru a crea userid.
7. Inserați următoarele două înregistrări în tabel, prin intermediul scriptului creat.
8. Verificați adăugările.
9. Salvați modificările. Ștergeți și modificați date în tabelul MY_EMPLOYEE.
10. Modificați numele de familie al angajatului cu ID = 3 în Drexler.
11. Modificați salariul la 1000 pentru toți cei cu salariul mai mic decât 900.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

12. Verificați modificările făcute.
13. Ștergeți-o pe Betty Dancs din tabelul MY_EMPLOYEE.
14. Verificați modificările.
15. Salvați toate modificările temporare. Controlați tranzacțiile de date asupra tabelului MY_EMPLOYEE.
16. Inserați în tabel ultima înregistrare rulând scriptul creat la punctul 6.
17. Verificați adăugarea.
18. Marcați un savepoint în cadrul tranzacției.
19. Ștergeți toate înregistrările din tabel.
20. Verificați că tabelul este gol.
21. Anulați cea mai recentă operație DELETE fără a anula și INSERT-ul anterior.
22. Verificați că rândul nou este intact. Salvați adăugarea, făcând-o permanentă.
23. Creați tabelul sal_history, tabelul mgr_history și tabelul special_sal cu structurile de mai jos:

sal_history

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
HIRE_DATE		DATE
SALARY		NUMBER(8,2)

mgr_history

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
MANAGER_ID		NUMBER(6)
SALARY		NUMBER(8,2)

special_sal

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
SALARY		NUMBER(8,2)

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

24. a. Scrieți o interogare care să execute următoarele:

- Să afișeze EMPLOYEE_ID, hire_date, salary, și manager_ID pentru acei angajați pentru care EMPLOYEE_ID este mai mic de 125 (din tabela EMPLOYEES).
- Dacă salariul este mai mare de 20,000, inserăți datele despre angajați (EMPLOYEE_ID și salary) în tabela SPECIAL_SAL.
- Inserăți valorile pentru EMPLOYEE_ID, hire_date, salary în tabela SAL_HISTORY.
- Inserăți valorile pentru EMPLOYEE_ID, manager_ID și salary în tabela MGR_HISTORY

b. Afisați rândurile din tabela SPECIAL_SAL.

EMPLOYEE_ID	SALARY
100	24000

c. Afisați rândurile din tabela SAL_HISTORY.

EMPLOYEE_ID	HIRE_DATE	SALARY
101	21-SEP-89	17000
102	13-JAN-93	17000
103	03-JAN-90	9000
104	21-MAY-91	6000
107	07-FEB-99	4200
124	16-NOV-99	5800

d. Afisați rândurile din tabela MGR_HISTORY.

EMPLOYEE_ID	MANAGER_ID	SALARY
101	100	17000
102	100	17000
103	102	9000
104	103	6000
107	103	4200
124	100	5800

6 rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

25. a. Creați tabela sales_source_data cu structura de mai jos și introduceți câteva datele prezentate în figură.

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
WEEK_ID		NUMBER(2)
SALES_MON		NUMBER(8,2)
SALES_TUE		NUMBER(8,2)
SALES_WED		NUMBER(8,2)
SALES_THUR		NUMBER(8,2)
SALES_FRI		NUMBER(8,2)

EMPLOYEE_ID	WEEK_ID	SALES_MON	SALES_TUE	SALES_WED	SALES_THUR	SALES_FRI
178	6	1750	2200	1500	1500	3000

- b. Creați tabela sales_info cu structura de mai jos:

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
WEEK		NUMBER(2)
SALES		NUMBER(8,2)

- c. Scrieți o interogare care să afișeze EMPLOYEE_ID, week_ID și vânzările din fiecare zi a săptămânii (din tabela SALES_SOURCE_DATA). Construiți o transformare astfel încât fiecare rând din tabela SALES_SOURCE_DATA să fie convertită în mai multe înregistrări pentru tabela SALES_INFO și afișați datele obținute.

EMPLOYEE_ID	WEEK	SALES
178	6	1750
178	6	2200
178	6	1500
178	6	1500
178	6	3000

5 rows selected.

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Capitolul 12 Anexa 1

Structura tabelelor folosite în carte și datele standard conținute de acestea

Tabela COUNTRIES

DESCRIBE countries

Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

SELECT * FROM countries;

CO	COUNTRY_NAME	REGION_ID
CA	Canada	2
DE	Germany	1
UK	United Kingdom	1
US	United States of America	2

Tabela DEPARTMENTS

DESCRIBE departments

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

SELECT * FROM departments;

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

Tabela EMPLOYEES

DESCRIBE employees

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

SELECT * FROM employees;

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000		103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200		103	60
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800		100	50
141	Trenna	Rajs	TRAJS	660.121.8009	17-OCT-95	ST_CLERK	3500		124	50
142	Curtis	Davies	CDAVIES	660.121.2994	29-JAN-97	ST_CLERK	3100		124	50
143	Randall	Matos	RMATOS	660.121.2874	15-MAR-98	ST_CLERK	2600		124	50
144	Peter	Vargas	PVARGAS	660.121.2004	09-JUL-98	ST_CLERK	2500		124	50
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	.2	100	80
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	.3	149	80
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600	.2	149	80
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	.15	149	
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400		101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000		100	20
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000		201	20
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000		101	110
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300		205	110

Tabela JOBS

DESCRIBE jobs

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

SELECT * FROM jobs;

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
AC_MGR	Accounting Manager	8200	16000
AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000

Tabela JOB_GRADES

DESCRIBE job_grades

Name	Null?	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

SELECT * FROM job_grades;

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Tabela JOB_HISTORY

DESCRIBE job_history

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

SELECT * FROM job_history;

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
102	13-JAN-93	24-JUL-98	IT_PROG	60
101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
101	28-OCT-93	15-MAR-97	AC_MGR	110
201	17-FEB-96	19-DEC-99	MK_REP	20
114	24-MAR-98	31-DEC-99	ST_CLERK	50
122	01-JAN-99	31-DEC-99	ST_CLERK	50
200	17-SEP-87	17-JUN-93	AD_ASST	90
176	24-MAR-98	31-DEC-98	SA_REP	80
176	01-JAN-99	31-DEC-99	SA_MAN	80
200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

Facultatea de Automatică și Calculatoare Iași
Baze de date – lucrări practice

Tabela LOCATIONS

DESCRIBE locations

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

SELECT * FROM locations;

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	CO
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

Tabela REGIONS

DESCRIBE regions

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

SELECT * FROM regions;

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

Bibliografie

1. Nancy Greenberg, Priya Nathan, Introduction to SQL Student Guide Volume 1 and 2, Oracle Corporation, 2001;
2. Visual FoxPro ®;
3. C. Botez, D. Arotăriței, I. Cârlig, D. Buzea, Baze de date – Lucrări de laborator, Editura Medex 2005, ISBN: 973-86741-0-7
4. A.Silberschatz, H.F. Korth, S.Sudarshan, Database System Concepts, Fourth Edition, McGraw-Hill Higher Education, ISBN: 0-07-112268-0,2002.