SQL

Universitatea Bucuresti

Colegiul de Informatica

LUCRARE DE DIPLOMA

S Q L

Structured Query Language

coordonator stiintific

ILEANA POPESCU

absolvent

MARIN GEORGE

Bucuresti - iunie - 1999

CUPRINS

3.0 Clauza where *

4.0 Operatori *

4.1 Operatori aritmetici *

4.1.1 Plus(+) *

4.1.2 Impartire(/) *

4.2 Operatori de comparatie *

4.3 Operatori caracter *

4.3.1 Operatorul LIKE *

4.3.2 Underscore(Subliniere)(_) *

4.3.3 Concatenarea (||) *

4.4 Operatori logici *

4.4.1 AND (SI) *

4.4.2 OR(SAU) *

4.4.3 NOT (nu) *

4.5 Operatorii pentru multimi *

4.5.1 Union si Union all *

4.5.2 INTERSECT *

4.6 Minus *

4.7 Alti operatori, IN si BETWEEN *

5.0 Functii din SQL *

5.1 Functii Totalizatoare *

5.1.1 COUNT *

5.1.2 SUM *

5.1.3 AVG *

5.1.4 MAX *

5.1.5 MIN *

5.1.6 VARIANCE *

5.1.7 STDDEV *

5.2 Functii pentru data calendaristica si ora *

5.2.1 ADD_MONTHS(adauga luni) *

5.2.2 LAST_DAY(ultima zi) *

5.2.3 MONTHS_BETWEEN(lunile dintre) *

5.2.4 NEW_TIME(Ora noua) *

5.2.5 NEXT_DAY(Ziua urmatoare) *

5.2.6 SYSDATE *

5.3 Functii aritmetice *

5.3.1 ABS *

5.3.2 CEIL si FLOOR *

5.3.3 COS,COSH,SIN,SINH,TAN,TANH *

5.3.4 EXP *

5.3.5 LN si LOG *

5.3.6 MOD *

5.3.7 POWER *

5.3.8 SIGN *

5.3.9 SQRT *

5.4 Functii caracter *

5.4.1 CHR *

5.4.2 CONCAT *

5.4.3 LENGTH *

5.5 Functii de conversie *

5.5.1 TO_CHAR *

5.6 Functii diverse *

5.6.1 USER *

6.0 Ordine in haos *

6.1 Clauza ORDER BY *

6.2 Clauza GROUP BY *

6.3 Clauza HAVING *

6.4 Combinatii de clauze *

7.0 JONCTIUNI *

7.1 Non/Echi –jonctiuni *

7.1.1 Echi-jonctiuni (jonctiuni echivalente) *

7.1.2 Non-echi-jonctiuni(jonctiuni neechivalente) *

7.2 JONCTIUNI EXTERNE *

7.2.1 Jonctiunea externa (Outer Join) *

7.2.2 Joncttiunea unui tabel cu el insusi(Autojonctiunea) *

8.0 SUBINTEROGARI *

8.1 Construirea unei subinterogari *

8.2 Folosirea functiilor totalizatoare in subinterogari *

8.3 Imbricarea subinterogarilor *

8.4 Subinterogari corelate *

9.0 Folosirea cuvintelor cheie exists, any si all *

10.0 Manipularea Datelor *

10.1 Instructiunea Insert *

10.1.1 Instructiunea Insert ...Values *

10.1.2 Instructiunea INSERT . . . SELECT *

10.2 Instructiunea UPDATE *

10.3 Instructiunea DELETE *

11.0 Crearea si intretinerea tabelelor *

11.1 Instructiunea CREATE DATABASE *

11.2 Proiectarea unei baze de date *

11.3 Crearea unui dictionar de date *

11.4 Crearea campurilor cheie *

11.5 Instructiunea CREATE TABLE *

11.5.1 Valoarea NULL *

11.6 Instructiunea ALTER TABLE *

11.7 Instructiunea DROP TABLE *

11.7 Instructiunea DROP DATABASE *

12.0 Crearea vederilor si a indecsilor *

12.1 Folosirea vederilor *

12.1.1 Restrictii la folosirea instructiunii SELECT *

12.1.3 Probleme care apar la modificarea datelor folosind vederile *

12.1.4 Aplicatii obisnuite ale vederilor *

12.1.5 Vederile si securitatea *

12.1.6 Instructiunea DROP VIEW *

12.2 Folosirea indecsilor *

16.8 Tabele asociative *

16.9 Algebra relationala *

16.9.1 Operatorul UNION *

16.9.2 Operarul DIFERENCE *

16.9.3 Operatorul INTERSECT *

16.9.4 Operatorul PROJECT *

16.9.5 Operatorul SELECT *

16.9.6 Operatorul DIVISION *

16.9.7 Operatorul PRODUCT *

16.9.8 Operatorul JOIN *

16.9.9 Operatorul NATURAL JOIN *

16.9.10 Operatorul q -JOIN *

16.9.11 Operatorul SEMI-JOIN *

16.9.12 Operatorul OUTER JOIN *

16.10 Expresii echivalente ale algebrei relationale *

Introducere in SQL

Istoria SQL incepe in laboratoarele IBM din San Jose, unde limbajul a fost dezvoltat in ultimii ani ai deceniului 8.Initialele provin de la Structured Query Language.

SQL este diferit de limbajele procedurale si 3GL.

SQL este un limbaj neprocedural. SQL este ceea ce face posibil un sistem de gestiune a bazelor de date relationale SGBDR (Relational Database Management System – RDBMS).Ceea ce diferentiaza un SGBD de un SGBDR este faptul ca ultimul asigura un limbaj de baze de date orientat pe multimi.Pentru majoritatea sistemelor de administrare a bazelor de date relationale, acest limbaj este SQL.

Structured Query Language (SQL - limbajul structurat de interogare) este limbajul styandart de facto folosit pentru manipularea si regasirea datelor din aceste baze de date relationale.Prin SQL, un programator sau un administrator de baze de date poate face urmatoarele lucruri:

sa modifice structura unei baze de date ;
sa schimbe valorile de configurare pentru securitatea sistemului;
sa adauge drepturi utilizatorilor asupra bazelor de date sau tabelelor;
sa interogheze o baza de date asupra unor informatii;
sa actualizeze continutul unei baze de date.
SQL a fost dezvoltat pentru a servi bazele de date relationale. Fara o intelegere minima a teoriei bazelor de date relationale, nu veti putea folosi efectiv SQL.

OBS: Voi folosi pe tot parcursul exemplelor diagrama conceptuala referitoare la activitatea unei agentii imobiliare (vezi anexa).

2.0 Blocurile componente ale operatiei de regasire a datelor: select si from

Vom observa ca pe masura ce experienta noastra in SQL creste vom folosi cuvintele SELECT si FROM mai des decat oricare alte cuvinte din vocabularul SQL.

Incepem cu comanda SELECT:

Tabelul AGENTI:

SINTAXA: SELECT <nume de coloana>

Intrare: SQL> select * from agenti;

Iesire:

cod_agent nume prenume sex data_nasterii adresa telefon comision

1 Bundy AL m 12/12/56 USA,IL,100 555-1111 25

2 MEZA AL f 10/01/65 UK,200 30

3 MERRICK BUD m 01/04/76 CO,300 555-6666 25

4 MAST JD m 03/11/77 LA,381 555-6767 25

Acest exemplu arata baza de date model .Semnul (*) transmite bazei de date comanda de a va returna toate coloanele asociate cu tabelul dat descris in clauza FROM.Ele sunt returnate intr-o ordine hotarata de baza de date.

2.1 Selectarea coloanelor individuale

Sa presupunem ca suntem interesati numai de o pereche de coloane.Introducem de la tastatura comanda:

Intrare: SQL> SELECT telefon,nume from agenti;

Iesire : telefon nume

Bundy
MEZA

MERRICK
MAST
Acum avem afisate coloanele care ne intereseaza. Observati folosirea literelor mari si mici in interogare. Acestea nu au nici un efect asupra rezultatului.

2.2 Interogari distincte

Daca privim cu atentie tabelul AGENTI vom observa ca unele date se repeta.

Cum procedam daca dorim sa vedem cate prenume diferite exista in coloana?

Sa incercam astfel:

Intrare: SQL> select DISTINCT prenume from agenti;

Iesire: prenume

AL

BUD

JD

Observam ca numai apare prenumele AL de doua ori. Deoarece am specificat

DISTINCT , este afisate numai o aparitie a prenumelui AL.

2.3 Expresii

Definitia unei expresii este simpla: aceasta returneaza o valoare.De asemenea, este un termen foarte larg, deoarece poate trata tipuri diferite, cum ar fi sir, numeric sau logic.

De fapt, aproape orice urmeaza o clauza (de exemplu, SELECT sau FROM) este o expresie, deci am folosit deja expresii.

In instructiunea : select (prenume + ‘ ‘ + nume) from agenti;

fraza (prenume + ‘ ‘ + nume) este o expresie care returneaza numele complet din baza de date AGENTI.O completare utila pentru aceasta ar putea fi

WHERE nume = ‘MAST’

care este mai interesanta.Contine conditia nume=’MAST’,care este un exemplu de expresie logica(nume=’MAST’ va fi falsa sau adevarata in functie de conditia =).

2.4 Conditii

Daca doriti vreodata sa gasiti un anumit element sau un grup de element in baza dumneavostra de date, aveti nevoie de una sau mai multe conditii (de asemenea clauza WHERE).In exemplul anterior , conditia este

nume = ‘MAST’

Daca problema dumneavoastra a fost sa gasiti toate persoanele din AGENTI care au numele mai mare decat ‘Bundy’ ati putea introduce conditia

nume >’Bundy’

Conditiile fac posibile interogarile selective.In cea mai obisnuita forma a lor, ele se compun dintr-o variabila, o constanta si un operator de comparare. In primul exemplu,

variabila este nume, constanta este ‘MAST’ si operatorul de comparare =. In al doilea exemplu , variabila este nume, constanat este ‘MAST’ si operatorul

de comparare este >. Mai avem nevoie de numai doua lucruri inainte de a incepe sa cream interogari conditionale: intelegerea operatorilor si de clauza WHERE.

3.0 Clauza where

Sintaxa clauzei WHERE este

WHERE <conditie de cautare>

Impreuna cu SELECT si FROM , WHERE este cea mai folosita clauza in SQL.Aceasta clauza va face interogarile mai selective, intr-un mod simplu. Fara clauza WHERE, cel mai util lucru pe care il puteti face cu o interogare este sa afisati baza de date.

Daca dorim o anume persoana , ati putea introduce comanda

Intrare: SQL> SELECT *

FROM agenti

WHERE nume =’Bundy’;

care ar genera o singura inregistrare:

Iesire:

cod_agent nume prenume sex data_nasterii adresa telefon comision

1 Bundy AL m 12/12/56 USA,IL,100 555-1111 25

Acest lucru pare interesant, sau cel putin, mai util.

4.0 Operatori

Operatorii sunt ceea ce dumneavoastra folositi in interiorul conditiei unei expresii pentru a specifica exact ceea ce doriti din baza de date. Operatorii se impart in sase grupe distincte : aritmetici, de comparare, caracter, logici, pentru multimi si alti operatori.

4.1 Operatori aritmetici

Operatorii aritmetici sunt plus(+), minus(-), impartire(/), inmultire(*) si modulo(%).

Primii patru sunt descrisi chiar prin numele lor. Ultimul, modulo, returneaza restul intreg al unei impartiri.De exmplu :

5 % 2 = 1

6 % 2 = 0

Operatorul modulo nu accepta tipuri de date care contin zecimale, ca de exemplu, tipul rreal sau number.

Daca plasati cativa operatori aritmetici intr-o expresie fara paranteze, ei sunt tratati in ordinea urmatoare: inmultire, impartire, module, adunare si apoi scadere.

In continuare avem un exemplu pentru operatorul plus(+).

4.1.1 Plus(+)

Semnul plus poate fi folosit in doua moduri diferite.

Mai intai afisam baza de date CURSVALUTAR.

Intrare: SQL> select *

from CURSVALUTAR;

Iesire:

tip_valuta data vanzare cumparare

USA $ 01/01/98 9540 10540

DM 01/01/98 5400 5550

ECU 01/01/98 12000 13000

Acum sa introducem comanda:

Intrare: SQL> select tip_valuta, data,vanzare, vanzare + 0.50

from cursvalutar ;

Iesire:

tip_valuta data vanzare vanzare + 0.10

USA $ 01/01/98 9540 10494

DM 01/01/98 5400 5940

ECU 01/01/98 12000 13200

Ce este ultima coloana? Nu face parte din tabelul original.Ce este cu acest titlu, deloc atragator, vanzare + 0.50?

SQL va permite crearea unor coloane virtuale sau derivate prin combinarea sau modificarea coloanelor existente.

Haideti sa modificam vanzare + 0.50 .

Introduceti comanda:

Intrare: SQL> SELECT tip_valuta, data, vanzare, (vanzare + 0.10) crestere _pret FROM cursvalutar;

Iesire:

tip_valuta data vanzare crestere_pret

USA $ 01/01/98 9540 10494

DM 01/01/98 5400 5940

ECU 01/01/98 12000 13200

Ati mai invatat ceva si anume alias – uri, iar daca va intrebati la ce va folosesc, cand vi se cere sa scrieti un generator de rapoarte amintiti-va de ceea ce ati invatat acum.

4.1.2 Impartire(/)

Semnul impartire are un singur mod clar de folosire.Utilizand tabela cursvalutar vom imparti campul vanzare la 2.

Intrare: SQL> select tip_valuta, data, (vanzare/2) test

from cursvalutar;

Iesire:

tip_valuta data vanzare test

USA $ 01/01/98 9540 4770

DM 01/01/98 5400 2700

ECU 01/01/98 12000 6000

4.2 Operatori de comparatie

Dupa cum semnifica si numele lor , operatorii de comparatie compara expresiile si returneaza una din urmatoarele trei valori: TRUE (Adevarat), FALSE (Fals) sau UnKnown (Necunoscut).In termenii folositi pentru baze de date, NULL semnifica absenta datelor dintr-un camp. Nu inseamna ca o coloana contine o valoare zero sau spatiu. un zero sau spatiu sunt valori.

NULL inseamna ca in campul respectiv nu se gaseste nimic. Daca faceti o comparatie de genul Field = 9 si Field este NULL, comparatia va returna UnKnown.

Observati tabelul AGENTI are o valoare NULL in campul telefon al persoanei MEZA.

Daca vreodata incercati sa gasiti o eroare logica de programare care pare sa nu aiba rezolvare, asigurati-va ca nu incercati sa comparati o valoare NULL si reveniti la valoarea prestabilita de FALSE.

Tabelul arata astfel:

Intrare: SQL> select *

from agenti;

Iesire:

cod_agent nume prenume sex data_nasterii adresa telefon comision

1 Bundy AL m 12/12/56 USA,IL,100 555-1111 25

2 MEZA AL f 10/01/65 UK,200 30

3 MERRICK BUD m 01/01/75 CO,300 555-6666 25

Haideti sa incercam sa gasim valoarea NULL:

Intrare: SQL> select *

from agenti

where telefon = null

Iesire: no rows selected.

Nu ati gasit nimic deoarece comparatia telefon = null returneaza valoarea false datorita faptului ca rezultatul este necunoscut. Acesta este un bun loc de folosire a unei valori

is null , modificand instructiunea where in WHERE telefon is null. In acest caz veti primi ca raspuns al comenzii toate liniile in care exista o valoare NULL.

Intrare: SQL> select *

from agenti

where telefon is null;

Iesire: cod_agent nume prenume sex data_nasterii adresa telefon comision

1 MEZA AL f 10/01/65 UK,200 30

4.3 Operatori caracter

4.3.1 Operatorul LIKE

Cum procedati daca vreti sa selectati parti ale unei baze de date care se potrivesc unui model dar nu foarte exact?

Ati putea folosi operatorul LIKE

Priviti urmatorul exemplu:

Intrare: SQL> select *

from agenti

where nume like ‘M%’;

Iesire:

cod_agent nume prenume sex data_nasterii adresa telefon comision

2 MEZA AL f 10/01/65 UK,200 30

3 MERRICK BUD m 01/04/76 CO,300 555-6666 25

4 MAST JD m 03/11/77 LA,381 555-6767 25

Veti primi ca raspuns toate inregistrarile care in care numele incepe cu M.

4.3.2 Underscore(Subliniere)(_)

Semnul underscore este un caracter de inlocuire pentru un singur caracter.

4.3.3 Concatenarea (||)

Simbolul || concateneaza doua siruri.

Urmatorul exemplu arata acest lucru:

Intrare: SQL> select nume || prenume nume_intreg

from agenti;

Iesire:

nume_intreg

Bundy AL

MEZA AL

MERRICK BUD

MAST JD

4.4 Operatori logici

4.4.1 AND (SI)

And inseamna ca ambele expresii intre care se gaseste operatorul trebuie sa fie adevarate pentru ca operatorul sa returneze valoarea TRUE. Daca oricare dintre ele este falsa, AND cva returna valoarea FALSE.

4.4.2 OR(SAU)

Daca vreuna din comparatii este TRUE, OR returneaza valoarea TRUE.

4.4.3 NOT (nu)

Not inseamna chiar ceea ce sugereaza numele.In cazul in care conditia aplicata este evaluta la valuarea true, operatorul NOT o schimba in false. In cazul in care conditia care urmeaza operatorului NOT are valoarea false, ea devine true.

Recomandare: Nu combinati operatorii not si or daca nu sunteti siguri de ceea ce reprezinta.

4.5 Operatorii pentru multimi

4.5.1 Union si Union all

UNION returneaza rezultatele a doua interogari, mai putin liniile duplicate.

Vom folosi ca baza de date model zone din Projectul Agentie Imobiliara.

Presupunem ca avem bazele de date ale doua agentii, una din Bucuresti (zone1), cealalta din Iasi(zone2).

Intrare:SQL> select *

from zone1;

UNION

select *

from zone2;

Iesire:

strada zona

Mihai Eminescu 21

Mihai Eminescu 32

Union all actioneaza exact ca UNION , cu exceptia faptului ca nu elimina duplicatele.

4.5.2 INTERSECT

Operatorul intersect returneaza numai liniile gasite de ambele interogari.

4.6 Minus

Minus (diferenta) returneaza liniile din prima interogare care nu fac parte din a doua .

4.7 Alti operatori, IN si BETWEEN

Cei doi operatori , IN si BETWEEN asigura folosirea unei forme scurte pentru functiile pe care stim deja sa le utilizam.

Obs: baza de date surse este din Projectul Agentie Imobiliara.

Exemplu:

Intrare: SQL> select *

from surse

where ziar in (’Adevarul’,’Romania Libera’, ‘Libertatea’);

Iesire:

nr_ziar ziar

1 Adevarul

2 Libertatea

11 Romania Libera

Intrare: SQL> select *

from zone

where zona between 13 and 20;

Iesire:

strada nr_zona

Aurel Botea 13

Laboratorului 18

Intrecerii 17

Heliade intre vii 16

Dristor 13

5.0 Functii din SQL

5.1 Functii Totalizatoare

Aceste functii se mai numesc functii de grup. Ele returneaza o valoare bazata pe valorile dintr-o coloana.

5.1.1 COUNT

Functia COUNT returneaza numarul de linii care respecta conditia din clauza WHERE.

Intrare: SQL> select COUNT(*) persoane_M

from agenti

where nume like ’M%’;

Iesire:

persoane_M

3

OBS: Nu este vreo diferenta daca folosim nume sau * in functia count pentru ca numai ce este implicat in clauza where are efect.

5.1.2 SUM

Functia SUM returneaza suma tuturor valorilor dintr-o coloana.

5.1.3 AVG

Functia AVG calculeaza valoarea medie a unei coloane.

5.1.4 MAX

Daca doriti sa gasiti cea mai mare valoare dintr-o coloana, folositi functia MAX.

Intrare:SQL> select MAX(zona)

from zone;

Iesire: MAX(zona)

40

Aceasta functie lucreaza atat cu numere cat si cu caractere.

5.1.5 MIN

MIN executa actiunea la care va asteptati si opereaza in acelasi mod ca MAX, cu exceptia faptului ca returneaza cea mai mica valoare dintr-o coloana.

5.1.6 VARIANCE

Aceasta functie are ca rezultat dispersia, adica patratul unei deviatii standart, un numar vital pentru multe calcule statistice.

5.1.7 STDDEV

Ultima functie de grup analizata, STDDEV, gaseste abaterea, sau deviatia standart a unei coloane de numere.

5.2 Functii pentru data calendaristica si ora

5.2.1 ADD_MONTHS(adauga luni)

Aceasta functie adauga un numar de luni la o data calendaristica specificata.

De exemplu: folosim tabelul cursvalutar

Intrare:SQL>select tip_valuta, data , ADD_MONTHS(data,2)

from cursvalutar;

Iesire:

tip_valuta data ADD_MONTH

USA $ 01-APR-98 01-JUN-98

DM 03-SEP-98 03-NOV-98

ECU 17-JAN-99 17-MAR-99

5.2.2 LAST_DAY(ultima zi)

Aceasta functie intoarce ca rezultat ultima zi a unei luni specificate.

5.2.3 MONTHS_BETWEEN(lunile dintre)

Daca doriti sa aflati cate luni sunt intre luna x si luna y, folositi functia MONTHS_BETWEEN.

5.2.4 NEW_TIME(Ora noua)

Daca doriti sa corectati ora in functie de ora teritoriului unde va aflati, aceasta este functia de care aveti nevoie.

5.2.5 NEXT_DAY(Ziua urmatoare)

Functia NEXT_DAY gaseste numele primei zile din saptamana egala sau ulterioara unei alte date calendaristice specificate.

5.2.6 SYSDATE

Functia SYSDATE returneaza ora si data calendaristica a sistemului.

5.3 Functii aritmetice

Multe din operatiile pe care doriti sa le efectuati asupra datelor dumneavoastra necesita folosirea matematicii.

5.3.1 ABS

Functia ABS returneaza valoarea absoluta a unui numar pe care il indicati.

5.3.2 CEIL si FLOOR

Prima dintre aceste functii, CEIL, returneaza cel mai mic numar intreg care este cel mai mare sau egal cu un argument.A doua, FLOOR, face exact operatia inversa, returnand cel mai mare numar intreg care este egal sau mai mic cu un argument.

5.3.3 COS,COSH,SIN,SINH,TAN,TANH

Functiile COS,SIN si TAN ofera baza de executie a numeroase operatii trigonometrice.

Functiile COSH,SINH si TANH returneaza valorile hiperbolice ale argumentelor primite ca parametri.Toate aceste functii opereaza presupunand ca argumentul de Intrare A are valoarea in radiani.

5.3.4 EXP

Daca aveti nevoie sa ridicati constanta e (constanta matematica folosita in diverse formule) la o putere, aceasta functie este exact ce va trebuie.

5.3.5 LN si LOG

Aceste doua functii se refera la algoritmi.Prima, LN, returneaza logaritmul natural al argumentului primit ca valoare de Intrare.

OBS: Sa nu uitam cumva sa precizam ca argumentul trebuie sa fie pozitiv!

Intrare:SQL>select A,LN(ABS(A))

from numere;

Iesire:

A LN(ABS(A))

1.1447004
-45 3.8066625

4 1.6094379

5.3.6 MOD

Ati intalnit functia MOD inainte de sectiunea prezenta.Ati vazut ca standartul ANSI pentru operatorul modulo, % , este implementat uneori ca functia MOD.

5.3.7 POWER

Pentru a ridica un numar la o putere, folositi functia POWER. In aceasta functie, primul argument este ridicat la puterea exprimata prin al doilea argument.

In aceasta functie daca primul argument este negativ, al doilea trebuie sa fie intreg.Folosim functia CEIL (sau FLOOR), ca in exemplul urmator:

Intrare:SQL> select A, CEIL(B), POWER(A,CEIL(B))

from numere;

Iesire: A CEIL(B) POWER(A, CEIL(B))

3.1415 4 97.3976

-45 1 -45

5 9 1953125

5.3.8 SIGN

Functia SIGN returneaza valoarea –1 daca argumentul primit este mai mic decat 0,

0 daca argumentul primit este egal cu 0 si 1 daca argumentul sau este mai mare decat 0.

5.3.9 SQRT

Functia SQRT returneaza radacina patrata a argumentului primit. Deoarece radacina patrata a unui numar negativ nu este definita, nu puteti folosi aceasta functie pentru numere negative.

5.4 Functii caracter

Multe dintre versiunile de SQL pun la dispozitie functii pentru manipularea caracterelor si a sirurilor de caractere.

5.4.1 CHR

Functia CHR returneaza caracterul echivalent cu numarul folosit ca argument.Caracterul este returnat in functie de setul de caractere al bazei de date (ex: tabelul ASCI).

5.4.2 CONCAT

Simbolul || lipeste doua randuri, la fel cum opereaza si functia CONCAT.

5.4.3 LENGTH

Functia LENGTH returneaza lungimea singurului argument caracter pe care il primeste.

5.5 Functii de conversie

Aceste functii va pun la dispozitie o metoda facila de convertire a unui tip de baza de data in alt tip.

5.5.1 TO_CHAR

Modul uzual de operare al acestei functii este conversia unui numar inr-un caracter.

Alte implementari ale limbajului folosesc aceasta functie pentru a converti in caracter alet tipuri de date, ca de exemplu, DATE, sau pentru a include diferite argumente de formatare.

5.6 Functii diverse

5.6.1 USER

Aceasta functie returneaza informatii despre sistem. USER returneaza numele utilizatorului curent al bazei de date.

6.0 Ordine in haos

6.1 Clauza ORDER BY

Din timp in timp, este absolut necesar ca rezultatele interogarii dumneavoastra sa fie prezentate intr-o anumita ordine.

Ex:

Intrare: SQL>select *

from agenti

ORDER BY nume;

Iesire:

cod_agent nume prenume sex data_nasterii adresa telefon comision

1 Bundy AL m 12/12/56 USA,IL,100 555-1111 25

2 MAST JD m 03/11/77 LA,381 555-6767 25

4 MERRICK BUD m 01/04/76 CO,300 555-6666 30

3 MEZA AL f 10/01/65 UK,200 30

6.2 Clauza GROUP BY

Clauza executa functia totalizatoare descrisa in instructiunea SELECT pentru fiecare din grupurile coloanei care urmeaza clauza GROUP BY.

Intrare:SQL>select nume,prenume,COUNT(prenume)

from agenti

GROUP BY prenume;

Iesire:

nume prenume count

Bundy AL 2

MERRICK BUD 1

MAST JD 1

Regula este sa nu folosim in instrunctiunea select coloane care au mai multe valori pentru aceeasi valoare a coloanei utilizate in clauza GROUP BY. Afirmatia inversa nu este corecta.Putem folosi in clauza GROUP BY coloane care nu sunt mentionate in instructiunea select.

Intrare:SQL>select nume, prenume, count (prenume)

from agenti

GROUP BY prenume;

Iesire: Dynamic SQL Error

-SQL error code = - 104

-invalid column reference

Aceasta afirmatie are sens atunci cand folosim pentru moment numai o componenta din SQL.

Intrare:SQL>select nume, prenume, sex

from agenti

where prenume=’AL’;

Iesire:

nume prenume sex

Bundy AL m

MEZA AL f

Amintiti-va ca aveti numai o linie pentru fiecare grup, atunci cand folositi clauza GROUP BY.

6.3 Clauza HAVING

Obs:: Clauza where nu lucreaza cu functii totalizatoare, deci avem nevoie de clauza HAVING.

Ex:Intrare:SQL>select data, avg(vanzare)

from cursvalutar

group by data

HAVING avg(vanzare)<10000;

Iesire: data AVG

01-APR-98 9547

03-SEP-98 5777

17-JAN-99 3433

Clauza HAVING va permite sa folositi funtii totalizatoare intr-o instructiune de comparare, asigurand pentru functiile totalizatoare ceea ce where asigura pentru linii individuale.

6.4 Combinatii de clauze

Intrare:SQL>select tip_valuta, data,vanzare

from cursvalutar

where tip_valuta=’USA$’

and data > 01/01/98

order by vanzare;

Iesire: tip_valuta data vanzare

USA $ 02/01/98 6540

USA$ 21/11/98 10400

USA$ 19/06/98 11000

7.0 JONCTIUNI

Una dintre cele mai puternice caracteristici ale SQL, este capacitatea de grupare (organizare) si manipulare a datelor din mai multe tabele. Folosind functia JOIN din SQL, puteti proiecta tabele mai mici si mai amanuntite (specifice), care sunt mult mai usor de intretinut.

Ex:

Intrare: SQL>select *

from zone, cursvalutar;

Iesire:

strada nr_zona tip_valuta data vanzare cumparare

Aurel Botea 13 USA $ 01/01/98 9540 10500

Aurel Botea 13 DM 01/01/98 5400 5700

Aurel Botea 13 ECU 01/01/98 12000 13000

Laboratorului 18 USA $ 01/01/98 9540 10500

Laboratorului 18 DM 01/01/98 5400 5700

Laboratorului 18 ECU 01/01/98 12000 13000

O examinare atenta a rezultatului primei dumneavoastra operatii JOIN ne arata ca fiecare linie din ZONE a fost adaugata la fiecare linie din CURSVALUTAR.

Aceasta jonctiune se numeste jonctiune CROSS (incrucisata).

Regula de baza este ca operatiile de jonctiune “lipesc” tabelele.

7.1 Non/Echi –jonctiuni

7.1.1 Echi-jonctiuni (jonctiuni echivalente)

Folosirea clauzei where pentru a selecta liniile combinate cu egalitatea dorita se numeste jonctiune echivalenta.

Exemplu:

Intrare:SQL>select o.tiploc, o.tipoferta, o.nrcamere, c.tiploc, c.tipcerere, c.nrcamere

from oferte o, cereri c

where o.tiploc=c.tiploc

and o.nrcamere=c.nrcamere;

Iesire:

tiploc tipoferta nrcamere tiploc tipcerere nrcamere

apartament inchiriere 3 apartament inchiriere 3

garsoniera vanzare 1 garsoniera cumparare 1

7.1.2 Non-echi-jonctiuni(jonctiuni neechivalente)

In timp ce jonctiunea echivalenta foloseste un semn = in instructiunea where, non-echi-jonctiunea foloseste orice alt semn in afara de =.

Intrare:SQL>select o.tiploc, o.tipoferta, o.nrcamere, c.tiploc, c.tipcerere, c.nrcamere

from oferte o, cereri c

where o.nrcamere < c.nrcamere;

Iesire:

tiploc tipoferta nrcamere tiploc tipcerere nrcamere

apartament inchiriere 2 vila inchiriere 5

garsoniera vanzare 1 apartament cumparare 3

7.2 JONCTIUNI EXTERNE

7.2.1 Jonctiunea externa (Outer Join)

Exemplu:

Intrare:SQL>select o.tiploc, o.tipoferta, o.nrcamere

from oferte o

RIGHT OUTER JOIN cereri C ON cereri.nrcamere=4;

Iesire:

tiploc tipoferta nrcamere tiploc tipcerere nrcamere

apartament inchiriere 4 vila inchiriere 4

null null null null null null

garsoniera vanzare 4 apartament vanzare 4

Aceasta expresie a determinat SQL sa returneze un set complet al tabelului din dreapta CERERI , si sa plaseze valori nule in campurile in care nrcamere <>4.

7.2.2 Joncttiunea unui tabel cu el insusi(Autojonctiunea)

Sintaxa acestei operatii este similara cu jonctiunea a doua tabele.

Exemplu:

Intrare:SQL>select *

from cursvalutar,cursvalutar;

Interogarea de mai sus produce acelasi numar de comparatii ca si jonctiunea a doua tabele de cate linii are baza de date cursvalutar.Acest tip de jonctiune ar putea fi util pentru verificarea corectitudinii interne a datelor.

Exemplu:

Intrare:SQL>select f.tip_valuta, f.data, f.vanzare, f.cumparare

s.tip_valuta, s.data, s.vanzare, s.cumparare

from cursvalutar f, cursvalutar s

where f.tip_valuta=s.valuta

and f.data<>s.data

La nivelul sau de baza o jonctiune produce toate combinatiile posibile ale liniilor prezente in fiecare tabel.

8.0 SUBINTEROGARI

8.1 Construirea unei subinterogari

Aceasta se construieste plasand o interogare in interiorul altei interogari.

Exemplu:

Intrare:SQL>select *

from agenti

where cod_agent in

(select cod_agent

from agenti

where prenume like “Geo%”);

Iesire:

cod_agent nume prenume sex data_nasterii adresa telefon comision

1 Marin George m 12/12/56 USA,IL,100 555-1111 25

2 Mihalache Georgiana m 03/11/77 LA,381 555-6767 30

8.2 Folosirea functiilor totalizatoare in subinterogari

Functiile totalizatoare SUM, COUNT, MIN, MAX si AVG returneaza fiecare dintre ele o singura valoare.

Intrare:SQL>select o.tiploc, o.tipoferta, o.nrcamere, c.tiploc, c.tipcerere, c.nrcamere

from oferte o, cereri c

where o.nrcamere < c.nrcamere

and (AVG(c.nrcamere) <5);

Iesire:

tiploc tipoferta nrcamere tiploc tipcerere nrcamere

apartament inchiriere 3 vila inchiriere 5

garsoniera vanzare 1 apartament cumparare 2

8.3 Imbricarea subinterogarilor

Imbricarea este operatia de lipire a mai multor subinterogari in serie, ca in exemplul urmator: select * frim ceva where

(subinterogare(subinterogare(subinterogare)));

Subinterogarile pot fi imbricate cat de adanc va permite versiunea dvs de SQL.

Intrare:SQL>select o.tiploc, o.tipoferta, o.nrcamere, c.tiploc, c.tipcerere, c.nrcamere

from oferte o, cereri c

where o.nrcamere < c.nrcamere

and (o.pret <=AVG(select c.pret from cereri));

Iesire:

tiploc tipoferta nrcamere tiploc tipcerere nrcamere

apartament inchiriere 2 vila inchiriere 5

garsoniera vanzare 1 apartament cumparare 3

8.4 Subinterogari corelate

Subinterogarile corelate va permit sa folositi o referinta externa, cu aceleasi rezultate.

Subinterogarea corelata semana foarte mult prin modul de actiune cu o jonctiune. Corelatia este stabilita prin folosirea unui element din interogare in subinterogare.

Intrare:SQL>select tiploc, confort, tipoferta, nrcamere

from oferte o

where ‘decomandat’ =

(select confort

from cereri c

where c.confort = o.confort);

Iesire:

tiploc confort tipoferta nrcamere

apartament decomandat inchiriere 3

garsoniera decomandat vanzare 1

9.0 Folosirea cuvintelor cheie exists, any si all

Folosirea cuvintelor cheie EXISTS, ANY si ALL nu este chiar atat de limpede pentru un observator oarecare. EXISTS preia o subinterogare ca argument si returneaza valoarea TRUE daca subinterogarea returneaza ceva si, respectiv, valoarea FALSE daca setul de rezultate este nul.

Intrare:SQL>select tiploc, confort, tipoferta

from oferte

where EXISTS

(select *

from oferte

where tipoferta = “vanzare”)

Iesire:

tiploc confort tipoferta

apartament decomandat vanzare

garsoniera decomandat vanzare

Observatie:

Folosirea instructiunii select * in subinterogare are loc in interiorul lui EXISTS. Prin

constructie, EXISTS nu este interesat de numarul de coloane returnat de interogare.

10.0 Manipularea Datelor

10.1 Instructiunea Insert

Instructiunea Insert (insereaza) se foloseste in operatia de introducere a datelor intr-o

baza de date. Poate fi separata in doua instructiuni:

INSERT . . . VALUE

si

INSERT . . .SELECT

10.1.1 Instructiunea Insert ...Values

Instructiunea INSER ... VALUES este folosita pentru introducerea datelor intr-un tabel,

cate o inregistrare la un moment dat. Este utila pentru operaii mici care presupun lucrul

cu doar cateva inregistrari.

Sintaxa : INSERT INTO nume_tabel

(col1, col2 . . .)

VALUES (valoare1, valoare2 . . .)

La inserarea datelor intr-un tabel prin folosirea acestei intructiuni, trebuie sa respectati

trei reguli :

Valorile folosite trebuie sa aiba exact acelasi tip de data ca si campurile in care sunt
adaugate.

Dimensiunea datei introduse trebuie sa fie mai mica de cat dimensiunea coloanei.De exemplu, un sir de 80 de caractere nu poate fi adaugat intr-o coloana de 40 de caractere.
Localizarea datei in lista VALUES trebuie sa corespunda locatiei din lista de coloane in care este adaugata.

Intrare:SQL>INSERT INTO zone

(nr_zona, strada)

VALUES (1999, ‘LUCRARE de DIPLOMA’);

Iesire: 1 rows created.

Pentru a va convinge singur ca aceasta instructiune opereaza cu adevarat, puteti executa o

simpla instructiune SELECT pentru a verifica inserarea noii inregistrari:

Intrare:SQL>select * from zone;

Iesire:

nr_zona strada

1 N.Balcescu

.....

LUCRARE de DIPLOMA
.....

10.1.1.1 Inserarea valorilor NULL

Atunci cand este creata o coloana, ii puteti atribui cateva caracteristici diferite. Una din

aceste caracteristici este aceea ca respectiva coloana ar putea (sau nu) avea capacitatea de

a contine valori NULL. O valoare NULL inseamna ca valoarea este vida. Nu este nici

zero, in cazul unui intreg, nici spatiu, in cazul unui sir. In locul acestora nu exista nici un

fel de data in coloana inregistrarii respective. Daca o coloana are definitia NOT NULL

(acestei coloane nu i se permite sa contina valori NULL), trebuie sa introduceti o

valoare pentru coloana respectiva atunci cand folositi instructiunea INSERT.

Instructiunea INSERT este anulata daca aceasta regula nu este respectata si apoi ar trebui

sa primiti un mesaj de descriere a erorii aparute.

10.1.2 Instructiunea INSERT . . . SELECT

Aceasta instructiunea permite utilizatorului sa copieze intr-un tabel informatii dintr-un alt

tabel sau grup de tabele. Deseori, pentru cresterea performantei, sunt create tabele de

cautare. Tabelele de cautare pot contine date care se intind pe mai multe tabele din mai

multe baze de date. Tabelele de cautare sunt memorate adesea pe masinile client/server

pentru a reduce traficul de retea.

Sintaxa unei instructiuni INSERT . . . SELECT :

INSERT INTO nume_tabel

(col1, col2, . . .)

select col1, col2, ...

where conditie_de_cautare

Este important de retinut ca Iesirea unei interogari standart de tip select devine apoi

Intrare intr-un tabel al unei baze de date.

Intrare:SQL> insert intro tabelnou (cod_oferta, codclient)

select cod_oferta, codclient

from oferte;

Puteti verifica executarea cu succes a operatiei asfel:

Intrare:SQL> select * from tabelnou;

Iesire:

cod_oferta cod_client

34
121
.....

Sunt cateva noi reguli care trebuie respectate la folosirea instructiunii INSERT ... SELECT:

Instructiunea select nu poate selecta linii din tabelul care a fost inserat.
Numarul de coloane din instructiunea INSERT INTO trebuie sa fie egal cu numarul de coloane returnate de instructiunea SELECT.
Tipurile de date ale coloanelor din instructiunea INSERT INTO trebuie sa fie
acelasi cu tipurile de date ale coloanelor returnate de instructiunea SELECT.

10.2 Instructiunea UPDATE

Instructiunea UPDATE (Actualizeaza) are acelasi scop, dar este folosita pentru modificarea valorilor din inregistrarile existente. Sintaxa acestei instructiuni este urmatoarea:

UPDATE nume_tabel

SET nume_coloana1 = valoare1

[, nume_coloane2 = valoare2] . . .

where conditie_de_cautare

Aceasta instructiune verifica in primul rand clauza WHERE. Pentru toate inregistrarile

din tabelul dat in care clauza WHERE este evaluata la valoarea TRUE, valoarea

corespunzatoare este actualizata.

Intrare:SQL>UPDATE zone

SET strada = ’LUCRARE de DIPLOMA - SQL’;

WHERE nr_zona = 1999;

Observatie: Daca din instructiunea UPDATE lipseste clauza WHERE, vor fi actualizate

toate inregistrarile din tabelul dat.

10.3 Instructiunea DELETE

La fel de des ca necesitatea de adaugare a datelor intr-o baze de date, apare si necesitatea

de stergere a altor baze de date pe baza unei anumite operatii din program. Sintaxa

instructiunii DELETE(Sterge) este urmatoarea:

DELETE FROM nume_tabel

WHERE conditie

Cateva lucruri pe care este bine sa le cunuastem atunci cand folosim clauza DELETE:

Instructiunea DELETE nu poate fi folosita pentru stergerea valorii dintr-un camp
individual (pentru aceasta, folositi instructiunea UPDATE). Instructiunea DELETE sterge inregistrari complete dintr-un singur tabel.

Ca si instructiunile INSERT si UPDATE, operatia de stergere a inregistrarilor
dintr-un singur tabel poate determina aparitia unor probleme de integritate referentiala in

cadrul altor tabele. Acest aspect trebuie retinut atunci cand modificati datele dintr-o baza

de date.

Prin folosirea instructiunii DELETE puteti sterge numai inregistrari din tabel, nu si tabelul. Pentru eliminarea unui intreg tabel se foloseste instructiunea DROP TABLE.
Intrare: SQL>DELETE from zone

where nr_zona <1999;

Iesire : 1998 rows deleted.

Crearea si intretinerea tabelelor

Instructiunea CREATE DATABASE

Atunci cand incepeti un proiect pentru o baza de date, prima etapa in administrarea datelor va fi intotdeauna crearea bazei de date.

Sintaxa tipica pentru instructiunea CREATE DATABASE arata astfel:

CREATE DATABASE nume_baza_de_date;

Proiectarea unei baze de date

Proiectarea corecta a unei baze de date este extrem de importanta pentru succesul aplicatiei dumneavoastra. In mod esential, normalizarea este procesul de “spargere” a datelor dumneavoastra in componente separate in vederea reducerii duplicatelor.Sunt mai multe niveluri de normalizare, fiecare dintre ele reducand repetitia datelor.

Intre factorii care pot influenta proiectul unei baze de date, sunt inclusi urmatorii:

Securitatea
Spatiul de disc disponibil
Viteza de cautare si de regasire a datelor
Viteza actulizarilor bazei de date
Viteza de realizare a unor jonctiuni multiple in vederea regasirii datelor
Suportul fisierelor temporare de catre SGBDR

Crearea unui dictionar de date

Dictionarul de date este cea mai importanta forma de documentatie pentru proiectantul de baze de date. Acest dictionar este folsit pentru:

Descrierea scopului bazei de date si a utilizatorilor.
Realizeaza documentatia bazei de date. Aceasta poate insemna oricare dintre urmatoarele specificatii : pe ce dispozitiv a fost creata ; dimensiunea prestabilita a bazei de date sau dimensiunea fisierului jurnal (folosit pentru a memora informatii despre operatiile efectuate cu baze de date in anumite SGBDR);
Includerea codului sursa SQL pentru oricare fisier scrip de instalare sau dezinstalare a instrumentelor pentru import/export.
Asigurarea descrierii amanuntite a fiecarui tabel din baza de date si a scopurilor acestora in contextul utilizarii lor.
Documentarea structurii interne a fiecarui tabel. Acesta poate include toate campurile si tipurile lor de date, cu comentarii, toti indecsii si toate vederile.
Includerea codului sursa SQL pentru toate procedurile rezidente si pentru toti declansatorii.
Asigurarea unei descrieri a cerintelor bazei de date, cum ar fi folosirea valorilor unice sau a valorilor NOT NULL. De asemenea, ar trebui mentionat daca aceste constrangeri sunt fortate la nivelul SGBDR sau daca programatorul de baze de date trebuie sa verifice aceste constrangeri in cadrul codului sursa.

Crearea campurilor cheie

Impreuna cu crearea documentatiei pentru proiectul bazei de date, cel mai important obiectiv in proiectarea bazei de date este separarea sructurii de tabele astfel incat fiecare tabel sa aiba o cheie primara si o cheie externa. Cheia primara ar trebui proiectata astfel incat sa indeplineasca urmatoarele cerinte:

Fiecare inregistrare este unica in cadrul unui tabel (nici o alta inregistrare din tabel nu are toate coloanele identice cu o alta inregistrare).
Pentru ca o inregistrare sa fie unica, este necesar ca toate coloanele sa fie unice. Aceasta inseamna ca este ideal ca datele dintr-o coloana sa nu se mai repete nicaieri in tabel.

Coloana care contine date unice in intregul tabel este cunoscuta sub numele de campul cheii primare. Un camp al cheii externe este un camp folosit pentru legarea unui tabel de cheia primara sau externa a unui alt tabel.

11.5 Instructiunea CREATE TABLE

Sintaxa de baza pentru instructiunea CREATE TABLE este urmatoarea:

CREATE TABLE nume_tabel

{ camp1 tip_de_data [NOT NULL],

camp2 tip_de_data [NOT NULL],

camp3 tip_de_data [NOT NULL] . . . }

Exemplu:

Intrare: SQL> CREATE TABLE zone (

nr_zona number,

strada char(30));

Aceasta instructiune creaza tabelul zone. In cadrul tabelului sunt doua campuri: nr_zona si strada.

11.5.1 Valoarea NULL

O valoare NULL este aproape un oximoron, deoarece detinerea unui camp ce contine o valoare NULL inseamna ca in campul respectiv nu este stocata nici o valoare.

La construirea unui tabel, majoritatea sistemelor de administrare a bazelor de date va permit sa atribuiti unei coloane cuvintele cheie NOT NULL .Aceasta inseamna ca in coloana respectiva nu pot fi memorate valori NULL pentru nici una din inregistrarile din tabel.

Intrare: SQL> CREATE TABLE zone (

nr_zona number NOT NULL,

strada char(30)) NOT NULL;

Observatie: O buna regula care ar trebui respectata este ca atat campului cheii primare, cat si tuturor campurilor cheilor externe sa nu li se permita sa contina valori NULL.

Instructiunea ALTER TABLE

Instructiunea ALTER TABLE permite administratorului de sistem sa schimbe structura unui tabel dupa crearea acestuia.

Comanda ALTER TABLE permite programatorului sa efectueze doua operatii:

sa adauge o coloana la un tabel existent.
sa modifice o coloana care deja exista.
Sintaxa instructiunii ALTER TABLE este urmatoarea:

ALTER TABLE nume_tabel

<ADD nume_coloana tip_data;

MODIFY nume_coloana tip_data;>

Exemplu:

Intrare:SQL>ALTER TABLE zone

add comentarii char(80);

Aceasta instructiune va adauga o coloana noua, cu numele COMENTARII, care poate memora pana la 80 de caractere. Noul camp va fi adaugat la dreapta campurilor existente.

Nu putem sa folosim instructiunea pentru a adauga sau sterge campuri dintr-o baza de date . Poate fi folosita pentru modificarea atributului NOT NULL al unei coloane in NULL, dar nu si in sens invers. Valoarea specificata a unei coloane poate fi modificata din NULL in NOT NULL numai daca nu sunt valori NOT NULL in coloana respectiva.

Ex:

Intrare: SQL>ALTER TABLE nume_tabel

MODIFY (nume_coloana tip_de _data NULL);

Instructiunea DROP TABLE
SQL pune la dispozitie o comanda pentru a elimina complet un tabel dintr-o baza de date. Executand comanda DROP TABLE , puteti sterge un tabel impreuna cu toate vederile asociate si indecsii corespunzatori. O data ce ati executat comanda nu mai puteti reveni asupra ei.

Intrare: SQL>DROP TABLE zone;

11.7 Instructiunea DROP DATABASE

Are un mod de folosire identic cu instructiunea DROP TABLE.

Sintaxa pentru aceasta instructiune este urmatoarea:

DROP DATABASE nume_baza_de_date;

12.0 Crearea vederilor si a indecsilor

12.1 Folosirea vederilor

Vederile (views) sunt, in esenta lor, tabele virtuale. Ele pot fi folosite pentru incapsularea interogarilor complete. Dupa crearea unei vederi pentru un set de date, puteti trata respectiva vedere ca pe orice alt tabel.

Sintaxa instructiunii CREATE VIEW este urmatoarea:

CREATE VIEW <numele_vederii> [(coloana1, coloana2, ...)] AS

SELECT <nume_tabel nume_de_coloane>

FROM <nume_tabel>

Intrare:SQL>CREATE VIEW zone_test AS

SELECT * FROM zone;

Pentru a avea confirmarea unei operatii reusite, puteti trata vederea ca pe orice tabel:

Intrare: SQL>SELECT * FROM zone_test;

Iesire:

nr_zone strada

N. Balcescu
.....

Instructiunea CREATE VIEW va permite, de asemenea, sa selectati coloane individuale dintr-un tabel si sa le plasati intr-o vedere.

12.1.1 Restrictii la folosirea instructiunii SELECT

SQL plaseaza cateva restrictii la folosirea instructiunii SELECT pentru a formula o vedere. Urmatoarele reguli sunt valabile atunci cand folositi instructiunea amintita:

Nu puteti folosi operatorul UNION
Nu puteti folosi clauza ORDER BY

12.1.2 Modificarea datelor folosind vederile

Intrare:SQL>create view zone_test as

select * from zone;

SQL>update zone_test;

set nr_zona = nr_zona * 2;

SQL>select * from zone;

Iesire:

nr_zone strada

N.Balcescu
4 Monumentului

6 M. Eminescu

.....

12.1.3 Probleme care apar la modificarea datelor folosind vederile

In continuare este prezentata o lista care contine cele mai obisnuite lucruri pe care trebuie sa le cunuoasteti atunci cand lucrati cu o vedere:

Instructiunile DELETE nu sunt permise in vederi ale tabelelor multiple.
Instructiunea INSERT nu este permisa decat daca toate coloanele cu atributul NOT NULL folosite in tabelul de baza sunt incluse la vedere. Aceasta se datoreaza faptului ca procesorul SQL nu cunoaste ce valori sa insereze intr-o coloana NOT NULL.
Daca inserati sau actualizati inregistrari intr-o vedere a unei combinari, toate inregistrarile care sunt actualizate trebuie sa apartina aceluiasi tabel fizic.
Daca folositi clauza DISTINCT pentru crearea unei vederi, nu mai puteti sa efectuati actualizari sau inserari de inregistrari in cadrul respective.
O coloana virtuala (o coloana care este rezultatul unui calcul sau al unei expresii) nupoate fi actualizata.

12.1.4 Aplicatii obisnuite ale vederilor

Vederile sunt folosite pentru a executa o serie de procese:

Executarea functiilor de securitate pentru utilizator.
Conversia intre diverse unitati de masura
Crearea unui format nou de tabel virtual
Simplificarea construirii interogarilor complexe.

12.1.5 Vederile si securitatea

Toate sistemele de baze de date relationale ofera o gama completa de servicii de securitate include chiar in sistem. In general, utilizatorii unui sistem de baze de date se impart pe grupuri, in functie de nivelul de folosire al bazei de date. Tipurile obisnuite de utilizatori sunt administratorul bazei de date (DataBase Administrator), dezvoltatorii bazei de date(DataBase Developers), personalul pentru introducere date(Data Entry Personnel) si utilizatorii publici(Public Users). Administratorul bazei de date va detine, probabil, controlul complet asupra sistemului, inclusiv drepturile de a folosi comenzile UPDATE, INSERT, DELETE si ALTER in baza de date. Grupul public va avea numai drepturi SELECT- si poate li se va permite sa foloseasca acest drept numai pentru anumite tabele din anumite baze de date.

12.1.6 Instructiunea DROP VIEW

Pentru orice comanda SQL de genul CREATE ... , exista si o comanda DROP ... .

CREATE VIEW si DROP VIEW nu fac exceptie de la aceasta regula.

Sintaxa este urmatoarea:

DROP VIEW nume_vedere;

Singurul lucru care trebuie retinut la folosirea comenzii DROP VIEW este ca toate vederile care faceau referire la vederea in cauza devin invalide.

12.2 Folosirea indecsilor

Indecsii pot fi folositi pentru prezentarea datelor intr-un format diferit de cel fizic in care datele sunt memorate datele pe disc. In plus, indecsii pot reordona datele memorate pe disc.

Indescii sunt folositi intr-o baza de date SQL din trei motive principale:

Pentru a intari constrangerile de integritate referentiala prin folosirea cuvantului cheie UNIQUE.
Pentru a facilita ordonarea datelor pe baza continutului campului indexat sau a altor campuri.
Pentru a optimiza viteza de executie a interogarilor

12.2.1 Ce sunt indecsii?

Datele pot fi regasite intr-o baza de date folosind doua metode. Prima metoda, adesea numita metoda de acces secvential (Sequential Access Method) cere ca SQL sa parcurga fiecare inregistrare pentru a le gasi pe cele care respecta conditia pusa. Este o metoda de cautare ineficienta, dar este singura cale prin care SQL poate localiza inregistrarea corecta.

Va reamintim ca SQL foloseste o structura arborescenta pentru a memora si regasi datele indecsilor. In varful arborelui, sunt memorati pointerii la grupurile de date. Aceste grupuri sunt numite noduri. In cadrul fiecaruia din aceste noduri, sunt memorati pointerii catre alte noduri. Nodurile care fac trimitere la valorile continute in partea stanga sunt mai mici decat valoarea nodului parinte. Pointerii la valorile continute in partea stanga sunt mai mari decat valoarea nodului parinte.

Sistemul bazei de date cauta prima data in nodul din varf pentru a gasi informatia cautata. Daca nu o gasestem urmeaza pur si simplu pointerii pana cand cautarea are succes.

Sintaxa de baza pentru crearea unui index este urmatoarea:

CREATE INDEX nume_index

ON nume_tabel (nume_coloana1, [nume_coloana2,], ...);

Intrare:SQL>create index index_id on oferte (codclient)

SQL>select cod_oferta, codclient from oferte;

Iesire:

cod_oferta codclient

1212
1 2344

2412
3576
.....

Acum tabelul OFERTE este sortat dupa campul codclient pana cand indexul va fi eliminat prin instructiunea DROP INDEX.

Ca de obicei instructiunea DROP este foarte directa:

DROP INDEX nume_index;

Atunci cand folositi un index, sistemul de baze de date creeaza un obiect index fizic (structura arborescenta) si poate refolosi indexul respectiv de fiecare data cand tabelul este suspus unei interogari.

12.2.2 Sfaturi pentru indexare

In continuare este prezentat o lista cu cateva sfaturi de care ar trebui sa va amintiti atunci cand folositi indecsi:

Pentru tabele mici, folosirea indecsilor nu aduce imbunatatiri de performanta.
Indecsii aduc mari imbunatatiri atunci cand coloanele dupa care efectuati operatia respectiva contin o diversitate mare de informatii sau mai multe valori NULL.
Indecsii pot optimiza interogarile dumneavoastra atunci cand aceste interogari returneaza o cantitate mica de date. Daca returnati printr-o interogare mai mult de atat, indecsii adauga pur si simplu timpi suplimentari de executie.
Indecsii pot creste viteza de regasire a datelor. Cu toate acestea, ei incetinesc actualizarea datelor. Amintiti-va acest aspect atunci cand realizati mai multe actualizari intr-o linie cu un index.
Indecsii ocupa spatiul din baza dumneavoastra de date. Daca folositi un sistem de gestiune a bazelor de date care va permite sa administrati spatiul de disc ocupat de baza de date, luati in considerare dimensiunea indecsilor in etapade planificare a dimensiunii bazei de date.
Indexati intotdeauna dupa campuri care sunt folosite in jonctiuni de tabele.Aceasta poate creste foarte mult viteza unei jonctiuni.
Majoritatea sistemelor de administrare a bazelor de date nu va permit crearea unui index dupa o vedere.
Nu indexati dupa campuri care sunt actualizate sau modificate regulat. Timpul suplimentar cerut pentru actualizarea constanta a indexului va va impiedica sa obtineti performanta pe care o doriti.

12.2.3 Indexarea dupa mai multe campuri

SQL va permite sa realizati indexari dupa mai multe campuri. Acest tip de index este numit index compus.

Intrare: SQL>create index index_id on oferte codclient, cod_oferta);

SQL> select cod_oferta, codclient from oferte;

Iesire:

cod_oferta codclient

1212
233 1212

4 2412

100 2412

.....

12.2.4 Indecsi si jonctiuni

Crearea unui index dupa un camp care este des folosit in jonctiuni poate optimiza considerabil performanta interogarii dumneavoastra.Totusi trebuie retinut ca in cazul crearii prea multi indecsi, acestia pot incetini performanta sistemului dumneavoastra, in loc sa o mareasca.

13.0 Controlul tranzactiilor

Controlul tranzactiilor, sau gestiunea tranzactiilor, se refera la capacitatea unui sistem de gestiune a bazelor de date relationale de a efectua tranzactii

intr-o baza de date. Tranzactiile sunt unitati de prelucrare care trebuie efectuate fie grupat, fie deloc. Prin unitate de prelucrare intelegem ca o tranzactie are un inceput si un sfarsit. Daca ceva gresit se petrece in timpul unei tranzactii, intreaga unitate de prelucrare poate fi anulata, dupa dorinta utilizatorului. Daca totul este corect,unitatea de prelucrare poate fi salvata in intregime in baza de date.

13.1 Inceperea unei tranzactii

In toate sistemele care accepta tranzacii, trebuie sa fie o cale de a indica explicit

sistemului ca incepeti tranzactia.

Sintaxa este urmatoarea:

SET TRANZACTION { READ ONLY | USE ROLLBACK SEGMENT segment}

Standardul SQL specifica destul de clar ca fiecare instructiune a SQL trebuie sa accepte o

consistenta a citirii la nivelul instructiunii. Aceasta inseamna ca datele trebuie sa fie

consistente in timpul executarii instructiunii.

Intrare: SQL>set tranzaction read only;

SQL>select * from agenti

where cod_agent = 5;

Executati alte operatii –

SQL>commit;

Optiunea SET TRANZACTION READ ONLY permite programatorului sa blocheze

efectiv un set de inregistrari inainte de incheierea tranzactiei. Optiunea READ ONLY

poate fi folosita impreuna cu urmatoarele comenzi:

SELECT

LOCK TABLE

ALTER SESSION

ALTER SYSTEM

Optiunea USE ROLLBACK SEGMENT segment este folosita pentru a transmite

sistemului Oracle ce segment din baza de date sa foloseasca pentru a derula in sens

invers operatiile cu spatiul de memorare.

13.2 Terminarea tranzactiilor

Pentru a termina o tranzactie in limbajul SQL din Oracle, veti folosi urmatoarea sintaxa

de comanda:

COMMIT [WORK]

[COMMENT ‘text’

[FORCE ‘text’ [,intreg]];

Comanda COMMIT salveaza toate modificarile efectuate in cursul unei tranzactii.

Deseori este mai bine sa executati o instructiune COMMIT inainte de a incepe o

tranzactie noua. Aceasta obisnuinta va poate asigura ca nu vor fi facute erori si nu vor fi

blocate tranzactii anteriorare.

13.3 Anularea tranzactiei

Dupa inceperea unei tranzactii, este efectuata de obicei o procedura de verficare a erorilor

pentru a determina daca tranzactia a fost executata cu succes pana in momentul respectiv.

SQL pune la dispozitia utilizatorilor instructiunea ROLLBACK chiar din acest motiv.

Instructiunea ROLLBACK readuce tranzactia la forma initiala. Aceasta inseamna ca

starea bazei de date este refacuta in forma care exista inainte de inceperea tranzactiei

Sintaxa pentru aceasta comanda folosind Oracle 7 este urmatoarea:

ROLLBACK [WORK]

[TO [SAVEPOINT] punct_de_salvare

[FORCE ‘text’]

Intrare: SQL>set tranzaction read only;

SQL>select * from agenti

where cod_agent = 5;

Executati alte operatii –
SQL>commit;

In cazul optiunii CANCEL:

SQL>ROLLBACK;

13.4 Folosirea punctelor de salvare in tranzactii

In unele cazuri, ati dori sa executati “pe jumatate” tranzactia dumneavoastra, adica sa

executati o parte dintre instructiuni. Pentru a realiza acest lucru, trebuie sa folositi un

punct de salvare.Toate instructiunile care au fost executate pana in acest punct de salvare

sunt salvate.

Sintaxa pentru crearea unui punct de salvare folosind Oracle SQL este urmatoarea:

SAVEPOINT nume_punct_de_salvare;

Intrare: SQL>set tranzaction read only;

SQL>select * from agenti

where cod_agent < 5;

SQL>SAVEPOINT salveaza;

SQL>delete from agenti where cod_agent =3;

SQL>ROLLBACK TO SAVEPOINT salveaza;

SQL>commit;

Iesire:

cod_agent nume prenume sex data_nasterii adresa telefon comision

1 Bundy AL m 12/12/56 USA,IL,100 555-1111 25

2 MAST JD m 03/11/77 LA,381 555-6767 25

3 MEZA AL f 10/01/65 UK,200 30

4 MERRICK BUD m 01/04/76 CO,300 555-6666 30

14.0 Personal Oracle 7 si securitatea bazelor de date

Oracle 7 implementeaza functia de securitate folosind trei structuri :

Utilizatori
Roluri
Privilegii(Drepturi)

14.1 Crearea utilizatorilor

Utilizatorii sunt nume de conturi carora li se permite sa se conecteze la baza de date Oracle. Sintaxa SQL folosita pentru a crea un nou utilizator este:

CREATE USER uitlizator

IDENTIFIED { BY parola | EXTERNALLY}

[DEFAULT TABLESPACE spatiu_de_tabel]

[TEMPORARY TABLESPACE spatiu_de_tabel]

[QUOTA {intreg [K | M | UNLIMITED } ON spatiu_de_tabel] . . .

[PROFILE profil]

Daca este aleasa optiunea BY parola, sistemul cere utilizatorului sa introduca o parola de fiecare data cand se conecteaza in sistem. Ca un

exemplu, creati un utilizator pentru dumneavoastra. Exemplul meu este urmatorul:

Intrare:SQL>CREATE USER Bryan IDENTIFIED BY CUTIGER;

Iesire: User created.

De fiecare data cand ma conectez in retea (folosind alias-ul “Bryan”), mi se cere sa introduc parola CUTIGER.

Daca a fost aleasa optiunea EXTERNALY, sistemul Oracle preia de la sistemul de operare de pe calculatorul dumneavoastra numele de utilizator si parola. Cand va conectati in sistem, sunteti conectati automat si la Oracle.

Asa cum este cazul cu fiecare comanda CREATE, exista si o comanda ALTER USER:

ALTER USER utilizator

[IDENTIFIED {BY parola |EXTERNALY}]

[DEFAULT TABLESPACE spatiu_de_tabel]

[TEMPORARY TABLESPACE spatiu_de_tabel]

[QUOTA {intreg [K | M] | UNLIMITED } ON spatiu_de_tabel] . . .

[PROFILE profile]

[DEFAULT ROLE { rol [, rol] . . .

| ALL [EXCEPT rol [, rol] . . .| NONE}]]

Aceasta comanda poate fi folosita pentru modificarea tuturor optiunilor utilizatorului, inclusiv parola si profilul. De exemplu, pentru a schimba parola lui Bryan, veti introduce comanda:

SQL>ALTER USER Bryan

2>IDENTIFIED BY ROSEBUD;

Iesire:User altered.

Pentru a sterge un utilizator, folositi simplu comanda DROP USER. Aceasta sterge Intrarea utilizatorului in baza de date a sistemului.

Sintaxa comenzii :

DROP USER nume_utilizator [CASCADE];

Daca este folosita optiunea CASCADE, toate obiectele aflate in proprietatea utilizatorului respectiv sunt eliminate din sistem impreuna cu contul utilizatorului. Daca nu este folosita optiunea CASCADE si utilizatorul specificat prin nume_utilizator mai detine inca o serie de obiecte, utilizatorul nu este eliminat din sistem.

14.2 Crearea rolurilor

Pentru a acorda un rol unui utilizator, folositi sintaxa urmatoare:

GRANT rol TO utilizator [WITH ADMIN OPTION];

Daca este folosita optiunea WITH ADMIN OPTION, utilizatorul poate acorda la randul sau roluri altor utilizatori.

Pentru a sterge un rol, folositi comanda REVOKE:

REVOKE rol FROM utilizator;

Oracle va permite inregistrarea sub unul din urmatoarele trei roluri:

Connect (Conectare)

Resource (Resurse)
DBA (DataBase Administrator – Administrator de base de date)

14.2.1 Rolul Connect

Rolul Connect poate fi numit si rolul de nivel Intrare. Dupa ce un utilizator primeste accesul la rolul Connect, i se pot acorda diferite privilegii care ii permit efectuarea unei serii de operatii cu o baza de date.

Intrare:SQL>GRANT CONNECT TO BRYAN;

Iesire:Grant succeded.

Rolul Connect da utilizatorului sa selecteze, sa insereze, sa actualizeze si sa stearga inregistrari din tabele care apartin celorlalti utilizatori (dupa ce li s-au acordat drepturile corespunzatoare). De asemenea, utilizatorul poate crea tabele, vederi, secvente, unitati, de alocare(clustere) si sinonime.

14.2.2 Rolul Resource

Rolul Resource da utilizatorului un acces mai mare la bazele de date Oracle. In plus fata de drepturile care pot fi acordate rolului Connect, rolurilor Resource li se pot acorda drepturilor de a crea proceduri, declansatori si indecsi.

Intrare:SQL>GRANT RESOURCE TO BRYAN;

Iesire:Grant succeded.

14.2.3 Rolul DBA

Rolul DBA include toate drepturile. Utilizatorii care au acest rol au posibilitatea de a face practic orice doresc in sistemul de baze de date. Numarul de utilizatori cu acest rol ar trebui sa fie redus la minimum pentru a

asigura integritatea sistemului.

Intrare:SQL>GRANT DBA TO BRYAN;

Iesire:Grant succeded.

Haideti sa eliminam celelalte doua roluri:

Intrare:SQL>REVOKE CONNECT FROM BRYAN;

SQL>REVOKE RESOURCE FROM BRYAN;

Drepturile in sistem sunt drepturi care se refera numai la sistem.Sintaxa folosita pentru acordarea unui drept in sistem este urmatoarea:

GRANT drept_in_sistem TO { nume_utilizator | rol | PUBLIC }

[WITH ADMIN OPTION]

Intrare:SQL>GRANT CREATE VIEW

TO PUBLIC;
Iesire: Grant succeded.

Drepturile in sistem sunt drepturi care se refera numai la sistem. Sintaxa folosita pentru acordarea unui drept in sistem este urmatoarea:

GRANT drept_in_sistem TO { nume_utilizator | rol | PUBLIC}

[WITH ADMIN OPTION]

Sa presupunem ca vreti sa permiteti fiecarui utilizator din sistem sa aiba acces la comanda CREATE VIEW in propriile lor scheme.

Intrare:SQL>GRANT CREATE VIEW

>TO PUBLIC;
Iesire: Grant succeded.

Cuvantul cheie PUBLIC se refera la faptul ca oricine are dreptul sa foloseasca instructiunea CREATE VIEW.

14.3 Tabel - Drepturi in sistem obisnuite pentru Oracle 7

Drept(privilegiu) in sistem Operatii permise
ALTER ANY INDEX Permite posesorului sa modifice orice index din orice schema.
ALTER ANY PROCEDURE Permite posesorului sa modifice orice procedura rezidenta, functie sau pachet din orice schema.
ALTER ANY ROLE Permite posesorului sa modifice orice rol din baza de date.
ALTER ANY TABLE Permite posesorului sa modifice orice tabel sau vedere din schema.
ALTER ANY TRIGGER Permite dreptul de a activa, dezactiva sau compila orice declansator de baza de date din orice schema.
ALTER DATABASE Permite posesorului sa modifice baza de date.
ALTER USER Permite posesorului sa modifice orice utilizator. Acest drept autorizeaza posesorul sa modifice parola si metoda de autentificare a oricarui utilizator, sa atribuie cote de spatiu pe disc oricarui spatiu pentru tabel, sa stabileasca spatii pentru tabele

implicite si temporare si sa atribuie un

profil si roluri prestabilite.

CREATE ANY INDEX Permite posesorului sa creeze un index in orice tabel din orice schema.
CREATE ANY PROCEDURE Permite posesorului sa creeze proceduri

rezidente, functii si pachete in orice

schema.

CREATE ANY TABLE Permite posesorului sa creeze tabele in

orice schema. Proprietarul schemei care

contine tabelul trebuie sa aiba cote de

spatiu in spatiul pentru tabele, pentru ca

tabelul sa poate fi inclus.

CREATE ANY TRIGGER Permite posesorului sa creeze un declansator pentru o baza de date in orice schema asociat cu un tabel din orice schema.
CREATE ANY VIEW Permite posesorului sa creeze vederi in orice schema.
CREATE PROCEDURE Permite posesorului sa creeze proceduri rezidente, functii si pachete in propria sa schema.
CREATE PROFILE Permite posesorului sa creeze profile.
CREATE ROLE Permite posesorului sa creeze roluri.
CREATE SYNONYM Permite posesorului sa creeze sinonime in propriile scheme.
CREATE TABLE Permite posesorului sa creeze tabele in propriile scheme. Pentru a crea un tabel, posesorul dreptului trebuie sa detina cote de spatiu in spatiul pentru tabel, necesar pentru includerea tabelului.
CREATE TRIGGER Permite posesorului sa creeze un declansator pentru o baza de date in propriile scheme.
CREATE USER Permite posesorului sa creeze utilizatori . Acest drept permite, de asemenea, atribuirea cotelor de spatiu in orice spatiu pentru tabele, configurarea spatiilor prestabilite si temporare pentru tabele si atribuirea unui profil ca parte a unei instructiuni CREATE USER.
CREATE VIEW Permite posesorului sa creeze vederi in propriile scheme.
DELETE ANY TABLE Permite posesorului sa stearga linii din tabele sau vederi in orice schema sau sa trunchieze tabele din orice schema.
DROP ANY INDEX Permite posesorului sa elimine indecsi din orice schema.
DROP ANY PROCEDURE Permite posesorului sa elimine proceduri rezidente, functii sau pachete din propriile scheme.
DROP ANY ROLE Permite posesorului sa elimine roluri.
DROP ANY SYNONYM Permite posesorului sa elimine sinonime particulare din orice schema.
DROP ANY TABLE Permite posesorului sa elimine tabele din orice schema.

DROP ANY TRIGGER Permite posesorului sa elimine declansatori pentru baze de date din orice schema.
DROP ANY VIEW Permite posesorului sa elimine noduri moduri de afisare din orice schema.
DROP USER Permite posesorului sa elimine utilizatori.
EXECUTE ANY PROCEDURE Permite posesorului sa execute proceduri sau functii(independente sau aflate in pachete) sau sa faca referire la variabile din pachete publice in orice schema.
TRANSACTION Permite refacerea tranzactiilor distribuite “in dubiu” in baza locala de date.
GRANT ANY PRIVILEGE Permite posesorului sa acorde orice drept in sistem.
GRANT ANY ROLE Permite posesorului sa acorde orice rol in baza de date.
INSERT ANY TABLE Permite posesorului sa insereze linii in tabele si vederi in orice schema.
LOCK ANY TABLE Permite posesorului sa blocheze tabele si vederi in orice schema.
SELECT ANY SEQUENCE Permite posesorului sa faca referire la secvente din orice schema.
SELECT ANY TABLE Permite posesorului sa interogheze tabele, vederi sau instantanee de date in orice schema.
UPDATE ANY Permite posesorului sa actualizeze linii in tabele.

14.4 Tabel – Drepturile asupra obiectelor permise in Oracle 7

ALL

ALTER

DELETE

EXECUTE

INDEX

INSERT

REFERENCES

SELECT

UPDATE

14.5 Instructiunea GRANT

GRANT {drept_asupra_obiectului | ALL [PRIVILEGESS]} [(coloana [, coloana] ...)]

[, { drept_asupra_obiectului | ALL [PRIVILEGESS]} [(coloana [, coloana] ...)] ...

ON [schema .] obiect

TO {utilizator | rol |PUBLIC }[, utilizator | rol |PUBLIC}] ...

[WITH GRANT OPTION]

14.6 Instructiunea REVOKE

REVOKE { drept_asupra_obiectului | ALL [PRIVILEGES]}

[, {drept_asupra_obiectului |ALL [PRIVILEGES]}]...

ON [schema .] obiect

FROM {utilizator | rol | PUBLIC} [, {utilizator | rol | PUBLIC}] ...

[CASCADE CONSTRAINTS]

Presupunem ca utilizatorul care a creat tabelul agenti este Bryan, de asemenea el are dreptul de a crea utilizatori.

Cream doi utilizatori: Jack si Jill.

Intrare:

SQL> create user Jack identified by Jack;

User created.

SQL>create user Jill identified by Jill;

User create.

SQL>grant connect to Jack;

Grant succeded.

SQL>grant resource to Jill;

Grant succeded.

Ii acordam lui Jack dreptul de a folosi comanda SELECT.

SQL>grant SELECT ON agenti TO Jack;

Grant succeded.

SQL>grant SELECT, UPDATE(comision) ON agenti TO Jill;

Grant succeded.

Pentru ca Jack sa poate selecta date din tabelul agenti, el trebuie sa adreseze tabelul respectiv impreuna cu numele utilizatorului care este proprietar al tabelului.

Intrare:SQL>select * from agenti;

Iesire: SELECT * FROM agenti

*

ERROR at LINE 1:

ORA-0042: table or view does not exist

Aici Jack a fost avertizat ca tabelul nu exista.

Intrare:SQL>select *

from Bryan.agenti;

In primul rand parasiti contul lui Jack si conectati-va din nou in sistem cu numele de

utilizator Jill.

Acum sa inseram o inregistrare noua in tabel:

Intrare:SQL>INSERT INTO Bryan.agenti

VALUES (1,”Morgan”,”Bryan”,”m”,12/12/60,

”N.Balcescu nr. 26”,8989989,30);

Iesire:

INSERT INTO Bryan.agenti

*

ERROR at line 1:

ORA-01031: insuficient privileges

Aceasta operatie nu a fost executata deoarece Jill nu are dreptul de acces INSERT asupra tabelului agenti.

Intrare:SQL>update Bryan.agenti

set nume=”JOE”

where cod_agent = 3;

Iesire:UPDATE BRYAN.agenti

*

ERROR at line 1:

ORA-01031: insuficient privilileges

Din nou, Jill a incercat sa depaseasca drepturile pe care le are. Natural, sistemul Oracle a intercepat eroarea si a corectat-o rapid.

Intrare:SQL>update Bryan.agenti

set comision = 50

where nume =”Bundy”;

Iesire:1 row updated.

Intrare:SQL>select * from Bryan.agenti;

Iesire:

cod_agent nume prenume sex data_nasterii adresa telefon comision

1 Bundy AL m 12/12/56 USA,IL,100 555-1111 50

2 MAST JD m 03/11/77 LA,381 555-6767 25

3 MEZA AL f 10/01/65 UK,200 30

4 MERRICK BUD m 01/04/76 CO,300 555-6666 30

Puteti vedea acum ca operatia de actualizare este efectuata doar atunci cand Jill nu isi depaseste drepturile care i-au fost acordate.

14.6 Folosirea sinonimelor in locul vederilor

Un sinonim opereaza exact ca o vedere.Sintaxa arata astfel:

CREATE [PUBLIC] SYNONYM [schema .] sinonim

FOR [schema .] obiect [@dblink]

15.0 SQL incapsulat

15.1 Ce inseamna exact SQL incapsulat?

Aceasta poate insemna scrierea procedurilor rezidente incapsulate in baza de date, care pot fi apelate de un program de aplicatie pentru executarea unor procese. Unele sisteme de baze de date contin kit-uri comlete de instrumente care va permit sa construiti obiecte ecran si meniu prin folosirea unei combinatii intre SQL si limbajul propriu de programare. Codul SQL este incapsulat in acest program combinat.

15.2 SQL Static si Dinamic

SQL Static se refera la folosirea instructiunilor SQL incapsulate direct in codul programului. Acest cod nu poate fi modificat in timpul executiei aplicatiei. De fapt, majoritatea variantelor de implementare a SQL Static necesita folosirea unui

pre-compilator care fixeaza instructiunile SQL in timpul executiei aplicatiei.

Sunt si avantaje, si dezavantaje ale folosirii SQL Static. Iata cateva avantaje:

Cresterea vitezei de executie
Verificarea erorilor in timpul compilarii
Dezavantajele folosirii acestei tehnici sunt urmatoarele:

Nu este flexibila.
Conduce la o dimensiune mare a codului (deoarece interogarile nu pot fi formulate in timpul executiei aplicatiei).
Codul nu este portabil pe alte sisteme de baze de date (un aspect care ar trebui luat permanent in considerare).
Pe de alta parte, SQL Dinamic permite programatorului sa construiasca o instructiune SQL in timpul executiei aplicatiei si sa transfere aceasta instructiune motorului bazei de date. Aceasta returneaza datele in variabilele de program, care sunt legate, de asemenea, in timpul executiei aplicatiei.

Exemplu SQL Static intr-o functie C:

```
BOOL Print_Employee_Info(void)

{

int Cod_Agent = 0;

char Nume[41] = “\0”;

char Adresa[81] == “\0”;

/* Acum vom lega fiecare camp pe care il vom selecta la o variabila de program */

#SQL BIND(cod_agent, Cod_Agent);

#SQL BIND(nume, Nume);

#SQL BIND(adresa, Adresa);
```

/* Instructiunile anterioare “leaga” campuri din baza de date cu variabile din program.

Dupa ce vom interoga baza de date, vom derula inregistrarile rezultate si le vom afisa pe ecran */

```
#SQL SELECT cod_agent, nume, adresa FROM agenti;

#SQL FIRST RECORD

if (cod_agent == NULL)

{

return FALSE;

}

while (cod_agent != NULL)

{

printf(“Cod_Agent = %d\n”,Cod_Agent);

printf(“Nume = %s\n”,Nume);

printf(“Adresa = %s\n”,Adresa);

#SQL NEXT_RECORD

}

return TRUE;

}
```

Dupa ce ati introdus codul si fisierul a fost salvat, programul este de obicei rulat cu ajutorul unui precomilator. Acest precompilator converteste liniile care incep cu directiva de compilare #SQL in cod C. Acest cod poate fi apoi compilat impreuna cu restul programului pentru a realiza procesul dorit.

16.0 ANEXA

16.1 Project Baze de Date – Agentie Imobiliara.Administrarea cererilor si ofertelor de apartamente, case, vile, terenuri intr-o Agentie Imobiliara.

16.2 DIAGRAMA ENTITATE – RELATIE

m (1) se ocupa de 1(1) 1(1) se ocupa de m(1)

OFERTE

cod_oferta

tiploc gresie negociabil

nrcamere faianta comision

confort parchet tv

stiloc gaze mobilat

etaj balcinchis tipproprietar

etajmax nrbalc alteinfo

totsuprafata nrgrsanitare tapet

sufrsup vechime conditionat

d1sup achitat liber

d2sup datavenirii eliberin

d3sup dataactualizarii tipoferta

d4sup nrbloc ofcheltuieli

basup scara

busup nrap

hol1sup codclient

hol2sup pret

vedere valuta

acces

telefon

AGENT

cod_agent

nume

prenume

sex

data_nasterii

adresa

telefon

comision

CERERI

cod_cerere

tiploc z4

nrcamere z5

confort altezone

stilloc conditionat

etajmin mobilat

etajmax nrbalc

etaj maxetajbloc

gresie parchet

faianta telefon

codclient tipcerere

pret crcheltuieli

valuta

datavenirii

dataactualizarii

tv

nrgrsanitare

z1

z2

z3

m(0) publicata in m(1) m(1) publicata in m(0)

SURSE

nr_ziar

ziar

ZONE

nr_zona

strada

CURSVALUTAR

tip_valuta

data

vanzare

cumparare

Tabelul SURSE are ca scop publicitatea in cadrul firmei , unde se publica oferta, cererea.

16.3 Cardinalitate Agent oferte, cereri

Un agent se poate ocupa de mai multe oferte sau cereri.

Cererea si oferta trebuie sa fie distribuita unui singur agent.

16.3.1 Cardinalitatea maxima

Cate oferte pot fi repartizate unui agent?

Multe.

Cati agenti se pot ocupa de o oferta ?

Unul.

Deci relatia agenti_se_ocupa_de_oferte are cardinalitatea maxima 1:m.

Cate cereri pot fi repartizate unui agent?

Multe.

Cati agenti se pot ocupa de o cerere ?

Unul.

Deci relatia agenti_se_ocupa_de_cereri are cardinalitatea maxima 1:m.

16.3.2 Cardinalitatea minima

Cate oferte trebuie sa fie repartizate unui agent?

Cel putin una.

Cati agenti trebuie sa se ocupe de o oferta ?

Cel putin unul.

Deci relatia agenti_se_ocupa_de_oferte cardinalitatea minima 1:1.

Cate cereri trebuie sa fie repartizate unui agent?

Cel putin una.

Cati agenti trebuie sa se ocupe de o cerere ?

Cel putin unul.

Deci relatia agenti_se_ocupa_de_cereri are cardinalitatea minima 1:1.

Mai multe oferte, cereri pot fi publicate in mai multe surse, iar mai multe surse pot avea mai multe oferte, cereri.

16.4 Cardinalitati oferte, cereri -> surse

16.4.1 Cardinalitatea maxima

Cate oferte pot fi publicate intr-o sursa de publicitate?

Multe.

In cate surse poate fi publicata o oferta?

Multe.

Deci relatia oferte_publicata_in_surse este o relatie many-many (n:m).

Cate cereri pot fi publicate intr-o sursa de publicitate?

Multe.

In cate surse poate fi publicata o cerere?

Multe.

Deci relatia cereri_publicata_in_surse este o relatie many-many n:m.

16.4.2 Cardinalitatea minima

Cate oferte trebuie sa fie publicate intr-o sursa de publicitate?

Zero.

In cate surse trebuie sa fie publicata o oferta?

Una.

Deci relatia oferte_publicata_in_surse are cardinalitatea minima 0:1.

Cate cereri trebuie sa fie publicate intr-o sursa de publicitate?

Zero.

In cate surse trebuie sa fie publicata o cerere?

Una.

Deci relatia cereri_publicata_in_surse are cardinalitatea minima 0:1.

O oferta are o singura zona in care se gaseste clientul.
O cerere are mai multe zone in care isi poate gasi oferta.

16.5 Cardinalitatea oferte,cereri -> zone

16.5.1 Cardinalitatea maxima

Cate oferte se pot regasi intr-o zona ?

Multe.

In cate zone poate fi o oferta ?

Una.

Deci relatia oferte_se_afla_in_zone are cardinalitatea maxima m:1.

Cate cereri se pot regasi intr-o zona ?

Multe.

In cate zone poate fi o cerere ?

Multe.

Deci relatia cereri_se_afla_in_zone are cardinalitatea maxima m:m.

16.5.2 Cardinalitatea minima

Cate oferte trebuie sa fie intr-o zona ?

Zero.

In cate zone trebuie sa fie o oferta ?

Una.

Deci relatia oferte_se_afla_in_zone are cardinalitatea minima 0:1.

Cate cereri trebuie sa fie intr-o zona ?

Zero.

In cate zone poate fi o cerere ?

Cel putin una.

Deci relatia cereri_se_afla_in_zone are cardinalitatea minima 0:1.

Cheia primara pentru BD SURSE este nr_ziar.
Tabelul ZONE este pentru cautare in zone o anumita strada si o impartirea Bucurestiului in zone .
Cheia primara este nr_zona.

Tabelul CURSVALUTAR indica cursul valutar al valutei intr-o zi oarecare.Cheia primara este tip_valuta.
Tabelul OFERTE are drept cheie primara cod_oferta .

16.6 Tabele simple

AGENT

Cod_agent nume prenume sex data_nasterii telefon adresa comision(%)
0025 Popescu Vasile m 01/05/75 6542312 Str. Monumentului 25

ZONE

strada zona

Monumentului 32

OFERTE

cod_oferta tiplocuinta tipoferta nrcamere …alte campuri
4021 vila inchiriere 2 ……..

CERERI

cod_cerere tiplocuinta tipcerere nrcamere alte campuri
3121 garsoniera vanzare 1 ……

SURSE

nr_ziar ziar
1 Orice ziar

CURSVALUTAR

tipvaluta data vanzare cumparare

USA 1$dolari 12/11/98 9450 10500

16.7 Diagrama conceptuala

AGENT

Cod_agent

nume

prenume

sex

data_nasterii

telefon

adresa

comision

CERERI

cod_cerere

tiploc

tipcerere

nrcamere altecampuri

se_ocupa_de

cod_agent

cod_cerere

se_ocupa_de

cod_agent

cod_oferta

OFERTE

cod_oferta

tiploc

tipoferta

nrcamere altecampuri

---

publicata_in cod_oferta

ofcheltuieli

nr_ziar
publicata_in

cod_cerere

crcheltuieli

nr_ziar

SURSA

nr_ziar

ziar

16.8 Tabele asociative

se_ocupa_de

cod_agent cod_cerere
…….. ……

se_ocupa_de

cod_agent cod_oferta

…….. ……

publicata_in

cod_cerere crcheltuieli
nr_ziar

…… ……. .....

publicata_in

cod_oferta ofcheltuieli
nr_ziar

…… ……. .....

se_afla_in

nr_zona cod_oferta
....

se_afla_in

nr_zona cod_cerere
....

16.9 Algebra relationala

16.9.1 Operatorul UNION

Reuniunea a doua relatii R si S este multimea tuplurilor apartinand fie lui R, fie lui S, fie ambelor.

Exemplu:

1) Numele, prenumele si sex –ul agentilor care au data nasterii > 12/01/73 si sexul masculin.

Intrare: SQL> SELECT nume, prenume, sex FROM agenti where data_nasterii >”12/01/73”

UNION

SELECT nume, prenume, sex FROM agenti where sex=”m”;

16.9.2 Operarul DIFERENCE

Diferenta a doua relatii R si S este multimea tuplurilor care apartin lui R, dar nu apartin lui S. Diferenta este o operatie binara necomutativa care permite obtinerea tuplurilor ce apar numai intr-o relatie.

16.9.3 Operatorul INTERSECT

Intersectia a doua relatii R si S este multimea tuplurilor care apartin si lui R si lui S.

Operatorul INTERSECT si DIFERENCE pot fi simulati in SQL, cu ajutorul optiunilor EXISTS, NOT EXISTS, IN, != ANY, in cadrul comenzii SELECT.

Exemplu:

1) Selectam codul clientului care exista si in BD cereri si in BD oferte.

Intersectie in SQL:

```
SQL> select cod_cerere

from cereri

where EXISTS

( select cod_oferta

from oferte

where cereri.codclient= oferte.codclient

)
```

Diferenta in SQL:

```
SQL> select cod_cerere

from cereri

where NOT EXISTS

( select cod_oferta

from oferte

where cereri.codclient= oferte.codclient

)
```

16.9.4 Operatorul PROJECT

Proiectia este o operatie unara care elimina anumite atribute ale unei relatii producand o submultime “pe verticala” a acesteia.Suprimarea unor atribute poate avea ca efect aparitia unor tupluri duplicate, care trebuie eliminate.

Exemplu:

Vizualizarea numelor, prenumelor si a sexului fara dubluri;
SQL> select DISTINCT nume, prenume, sex

from agenti;

16.9.5 Operatorul SELECT

Selectia este o operatie unara care produce o submultime pe “orizontala” a unei relatii R.

Aceasta submultime se obtine prin extragerea tuplurilor din R care satisfac o conditie specificata.

Exemplu:

Vizualizarea tuturor tuplurilor din agenti care satisfac conditia sex=”f”;
SQL>SELECT *

FROM agenti

WHERE sex=”f”;

16.9.6 Operatorul DIVISION

Diviziunea este o operatie binara care defineste o relatie ce contine valorile atributelor dintr-o relatie care apar in toate valorile atributelor din cealalta relatie.

Operatorul DIVISION este legat de cuantificatorul universal (") care nu exista in SQL.

Cuantificatorul universal poate fi insa simulat cu ajutorul cuantificatorului existential ($) care se gaseste in limbajul SQL.

Prin urmare, operatorul DIVISION poate fi exprimat in SQL prin succesiunea a doi operatori NOT EXISTS.

Exemplu:1) Selectam codul unic al clientului mai mare de 100 care apare si in cerere, si in oferte.

```
SQL> select UNIQUE(codclient)

from cerere, oferte

where NOT EXISTS

(SELECT *

from cerere

where codclient >=100;

AND NOT EXISTS

(SELECT *

from cerere, oferte

where oferte.codclient=cereri.codclient));
```

16.9.7 Operatorul PRODUCT

Produsul cartezian al lui R cu S este multimea tuplurilor de aritate m + n unde primele m componente formeaza un tuplu in R, iar ultimele n componente formeaza un tuplu in S.

Exemplu:

Vizualizam tuplurile din oferte si cereri
SQL> select *

from cerere, oferte;

16.9.8 Operatorul JOIN

Operatorul de compunere permite regasirea informatiei din mai multe relatii corelate.Operatorul combina produsul cartezian, selectia si proiectia.

16.9.9 Operatorul NATURAL JOIN

Operatorul combina tupuri din doua relatii R si S, cu conditia ca atributele comune sa aiba valori identice. Compunerea naturala se noteaza prin JOIN(R, S) sau R S.

Exemplu:

Selectam tuplurile nrcamere, stiloc, confort din oferte si cereri;
SQL> SELECT o.nrcamere, o.stiloc, o.confort, c.nrcamere, c.codclient
from oferte o, cereri c

WHERE o.nrcamere=c.nrcamere;

Iesire:

nrcamere stiloc confort codclient nrcamere

3 decomandat 2 3232

1 comandat 1 5633

2 comandat 1 10321

16.9.10 Operatorul q -JOIN

Varianta in care se combina tupluri din 2 relatii cu conditia ca valorile atributelor specificate sa satisfaca o conditie JOIN(R, S, conditie) = s conditie(R x S) .Operatia reprezinta o varianta generalizata a operatiei de compunere naturala.

Exemplu:

Selectam tuplurile nrcamere, codclient, stiloc confort din oferte si cereri;
SQL> select o.nrcamere, o.stiloc, o.confort, c.nrcamere, c.codclient
from oferte c, cereri c

WHERE o.nrcamere <= 4;

Iesire: nrcamere stiloc confort codclient nrcamere

3 decomandat 2 3232 2

decomandat 2 4324 2
1 comandat 1 5633 1

2 comandat 1 10321 4

4 decomandat 2 14242 3

16.9.11 Operatorul SEMI-JOIN

Operatorul conserva atributele unei singure relatii participante la compunere si este utilizata doar daca sunt necesare atributele unei relatii. Operatorul este asimetric si se noteaza prin SEMIJOIN(R, S, conditie).Operatia de semi-compunere genereaza o relatie care contine toate tuplurile din R, corelate cu oricare din tuplurile lui S.

Exemplu:

Selectam tuplurile din oferte care au pretul din cereri < 10.000$
SQL > select *
from oferte

where cereri.pret<10000;

16.9.12 Operatorul OUTER JOIN

Operatorul atribuie valoarea null valorilor atributelor care exista intr-un tuplu din una din cele doua relatii de intrare, dar nu exista in a doua.

Exemplu:

SQL>select o.cod_oferta, o.nrcamere, o.tipoferta
c.cod_cerere, c.tipcerere,c.codclient

from cereri c

OUTER JOIN oferte o on oferte.codclient = 3320;

Iesire:

cod_oferta nrcamere tipoferta codclient

1922 3 inchiriere 3320

cod_cerere tipcerere codclient

null null null

234 inchiriere 3320

16.10 Expresii echivalente ale algebrei relationale

Exemplu 1: Numele, prenumele agentului cu codul >13 de sex feminin din baza de date agenti.

A1. R1=SELECT(nume, prenume, cod_agent >13)

R2=SELECT(nume, prenume, sex=F)

R3=INTERSECT(R1, R2)

Rezultat=R3

A2.Rezultat=(INTERSECT(SELECT(nume,prenume,cod_agent=13),

SELECT(nume,prenume, sex=F));

A3.Rezultat=P nume,prenume (s cod_agent > 13 (nume, prenume) Ç (s sex=’F’ (nume, prenume))

In SQL>select nume, prenume

from agenti

where cod_agent >13 and sex=F;

Exprimarea cererilor de date sub forma unor expresii din algebra relationala are la baza echivalenta dintre calculul relational si algebra relationala. Procesul de transformare

a cererilor se realizeaza conform unei strategii de optimizare care poate fi independenta de modul de memorare a datelor (strategie generala) sau dependenta de modul de memorare (strategie specifica unui anumit SGBD). Vom prezenta proprietatile algebrei relationale care permit implementarea strategiilor generale de optimizare.

Proprietatea 7: Comutarea proiectiei cu reuniunea

p cond(R1 È R2) = p cond(R1) È p cond(R2)

R1 = select(cod_agent, comision >=30)

R2 =select (cod_agent, nume like ‘M%’)

p (s cod_agent=comision >=30 È s cod_agent=”M%”) = p (s cod_agent=comision >=30) È p (s cod_agent=”M%”);

Proprietatea 8: Comutarea selectiei cu diferenta:

p cond(R1 - R2) = p cond(R1) - p cond(R2)

R1 = select(cod_agent, comision >=30)

R2 =select (cod_agent, nume like ‘M%’)

p (s cod_agent=comision >=30 - s cod_agent=”M%”) = p (s cod_agent=comision >=30) - p (s cod_agent=”M%”);

Proprietatea 3 Compunerea proiectiilor:

A Î B

p A (p B (R)) = p A (R)

p A (s cod_agent=nume like”M%”) (p B (s cod_agent=nume like “M%” or prenume like “M%”)) = p A (s cod_agent=nume like”M%”)

Proprietatea 4 Compunerea selectiilor

s cond1 (s cond2 (R)) = s cond1Ù cond2(R) = s (cond2(s cond1(R))

s cod_agent=comision >=30 (s cod_agent=nume like “A%”)) = s cod_agent(comision >=30 and nume like “A%”)