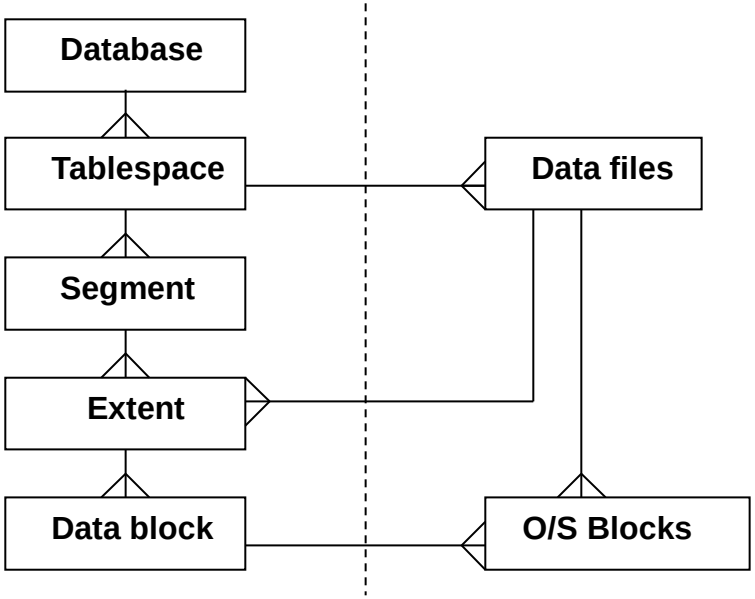
Limbajul SQL

Alexandru Boicea Curs: Baze de date

Structura bazei de date

- Structura logica si fizica a unei **Baze de Date** (*database*) relationale:



- Tablespace - este spatiul logic in care se creeaza obiectele (tabele, view-uri, indecsi, proceduri, etc.). O baza de date poate avea mai multe tablespace-uri, iar un tablespace poate avea alocat fizic mai multe fisiere de date (*data files*) .

Structura bazei de date

- Segmentul(*segment*) – reprezinta un spatiu logic de stocare alocat unui obiect intr-un tablespace. Pot fi de mai multe tipuri de segmente: permanente, temporare, index, rollback, etc.).
- Extensia(*extent*) - reprezinta o extensie logica a spatiului de stocare reprezentata printr-un numar continuu de blocuri.
- Blocul de date(*data block*) - reprezinta cea mai mica unitate logica de stocare.
- Fisiere de date(*data files*) - sunt fisierele organizate fizic pe un dispozitiv de stocare. Fisierele de date stocheaza fizic datele in baza de date.
- O/S block - reprezinta cea mai mica unitate fizica de organizare a datelor intr-o baza de date.

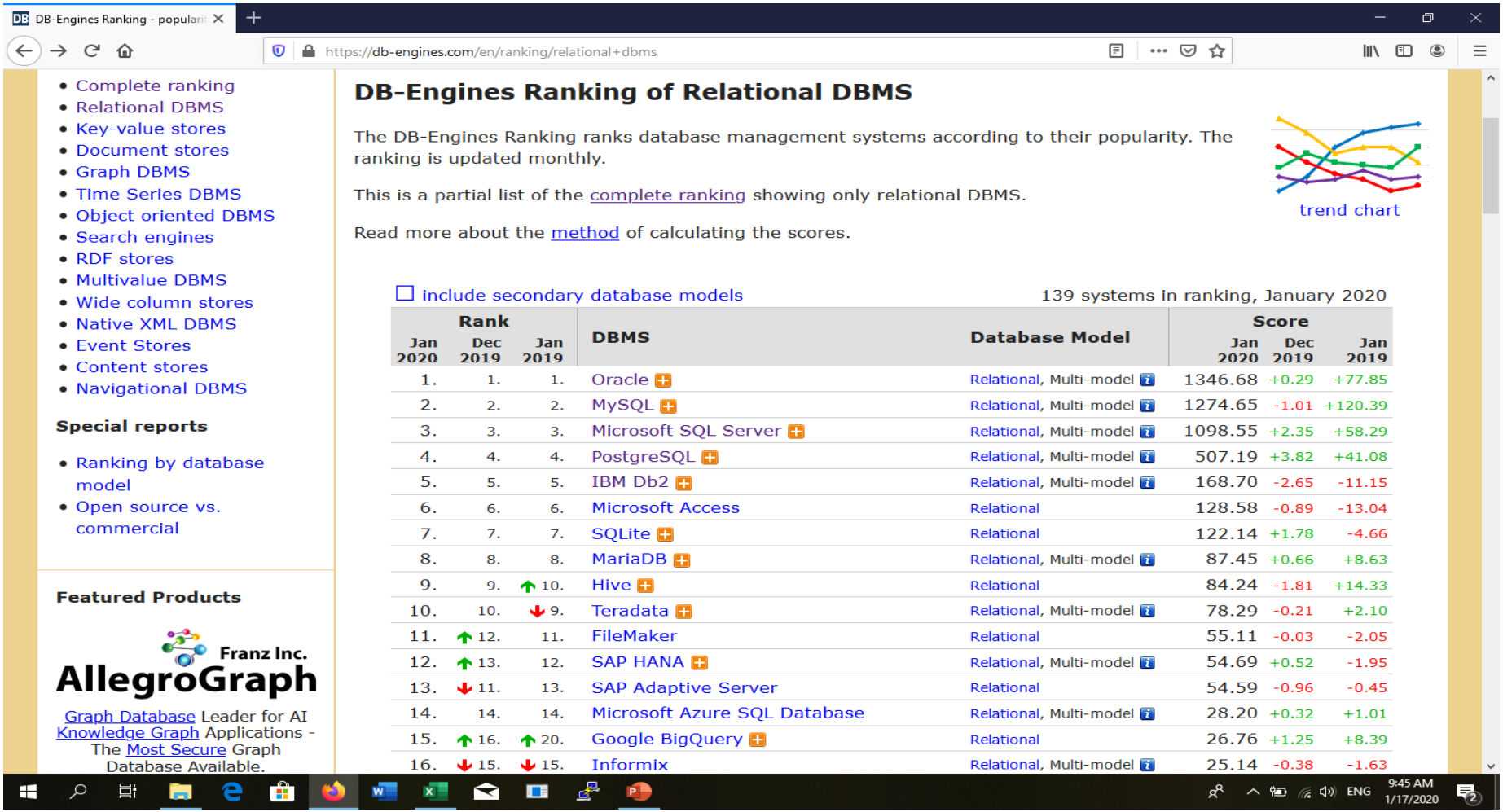
Sistemul de Gestiune a Bazei de Date

- Controlul asupra bazei de date este gestionat de catre **Sistemul de Gestiune a Bazei de Date**(SGBD) si verifica respectarea unor reguli:
  - O baza de date relationala apare ca o colectie de tabele definite de catre utilizator;
  - Utilizatorul nu controleaza felul cum este organizata fizic informatia;
  - controlul asupra fisierelor de date este gestionat exclusiv de catre sistemul de gestiune;
  - Utilizatorul poate defini anumiti parametri de sistem pentru optimizarea aplicatiilor sau pentru diferite setari;
  - Accesul la baza de date este gestionat exclusiv de catre sistem prin executarea de comenzi specifice;
  - Rularea aplicatiilor, atat pe server cat si pe masina client, este gestionata exclusiv de catre sistemul de gestiune.

Limbajul SQL

- Un sistem de gestiune a bazei de date necesita un limbaj de interogare pentru a permite utilizatorului sa acceseze datele.
- **Limbajul SQL** (*Structured Query Language*) este un limbaj de interogare structurat utilizat de majoritatea sistemelor de baze de date relationale.
- Cateva trasaturi caracteristice ale limbajului SQL:
  - Implementeaza setul standard de comenzi de manipulare a datelor(inserare, interogare, modificare, stergere);
  - Este un limbaj neprocedural care optimizeaza automat planul de executia a cererilor;
  - Cererile se executa secvential, linie cu linie, deci se prelucreaza o singura inregistrare dintr-o tabela la un moment dat .
  - Permite importul si exportul datelor;
  - Ofera suport pentru administrarea bazei de date.

Limbajul SQL



DB-Engines Ranking - popular

https://db-engines.com/en/ranking/relational+dbms

- Complete ranking
- Relational DBMS
- Key-value stores
- Document stores
- Graph DBMS
- Time Series DBMS
- Object oriented DBMS
- Search engines
- RDF stores
- Multivalue DBMS
- Wide column stores
- Native XML DBMS
- Event Stores
- Content stores
- Navigational DBMS

Special reports

- Ranking by database model
- Open source vs. commercial

Featured Products

Franz Inc. AllegroGraph

Graph Database Leader for AI Knowledge Graph Applications - The Most Secure Graph Database Available.

DB-Engines Ranking of Relational DBMS

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

This is a partial list of the complete ranking showing only relational DBMS.

Read more about the method of calculating the scores.

trend chart

include secondary database models

139 systems in ranking, January 2020

Rank Jan 2020	Rank Dec 2019	Rank Jan 2019	DBMS	Database Model	Score Jan 2020	Score Dec 2019	Score Jan 2019
1.	1.	1.	Oracle	Relational, Multi-model	1346.68	+0.29	+77.85
2.	2.	2.	MySQL	Relational, Multi-model	1274.65	-1.01	+120.39
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1098.55	+2.35	+58.29
4.	4.	4.	PostgreSQL	Relational, Multi-model	507.19	+3.82	+41.08
5.	5.	5.	IBM Db2	Relational, Multi-model	168.70	-2.65	-11.15
6.	6.	6.	Microsoft Access	Relational	128.58	-0.89	-13.04
7.	7.	7.	SQLite	Relational	122.14	+1.78	-4.66
8.	8.	8.	MariaDB	Relational, Multi-model	87.45	+0.66	+8.63
9.	9.	↑ 10.	Hive	Relational	84.24	-1.81	+14.33
10.	10.	↓ 9.	Teradata	Relational, Multi-model	78.29	-0.21	+2.10
11.	↑ 12.	11.	FileMaker	Relational	55.11	-0.03	-2.05
12.	↑ 13.	12.	SAP HANA	Relational, Multi-model	54.69	+0.52	-1.95
13.	↓ 11.	13.	SAP Adaptive Server	Relational	54.59	-0.96	-0.45
14.	14.	14.	Microsoft Azure SQL Database	Relational, Multi-model	28.20	+0.32	+1.01
15.	↑ 16.	↑ 20.	Google BigQuery	Relational	26.76	+1.25	+8.39
16.	↓ 15.	↓ 15.	Informix	Relational, Multi-model	25.14	-0.38	-1.63

ENG 9:45 AM 1/17/2020

https://db-engines.com/en/ranking/relational+dbms

Limbajul SQL

- In acest capitol vom face o introducere in limbajul **Oracle SQL** utilizat pentru accesarea si administrarea unei baze de date Oracle. Comenzile si cererile SQL sunt folosite pentru :
  - Inserarea/extragerea/modificarea/stergerea randurilor intr-o tabela;
  - Crearea/modificarea/stergerea obiectelor din baza de date;
  - Controlul conexiunii si accesului la baza de date;
  - Prelucrarea datelor;
  - Relationarea datelor din mai multe tabele;
  - Formatarea datelor de iesire;
  - Introducerea criteriilor de cautare si sortare a datelor;
  - Refacerea starii bazei de date la un moment anterior;
  - Importul, exportul si replicarea datelor.

Limbajul SQL

- ✓ Setul de comenzi standard SQL de manipulare a datelor **DML** (*Data Manipulation Language*) :
- SELECT – folosita pentru extragerea datelor din baza de date;
- INSERT – folosita pentru inserarea datelor in baza de date;
- UPDATE – folosita pentru modificarea datelor in baza de date;
- DELETE – folosita pentru stergerea inregistrarilor.
- ✓ Setul de comenzi standard SQL pentru definirea datelor **DDL** (*Data Definition Language*):
- CREATE – folosita pentru crearea unui obiect (tabela, view, index, etc.) in baza de date ;
- ALTER – folosita pentru modificarea structurii unui obiect din baza de date;
- DROP – folosita pentru stergerea unui obiect din baza de date.

Limbajul SQL

- ✓ Setul de comenzi standard SQL pentru crearea/revocarea drepturilor de acces:
- GRANT – folosita pentru a grantifica drepturile de acces la un obiect din baza de date;
- REVOKE – folosita pentru revocarea drepturilor de acces.

- Acestea sunt numai o parte a comenzilor SQL(lista completa se gaseste in *Manualul de Referinta a Limbajului SQL*).
- Cateva reguli de scriere a comenzilor SQL :
  - Comenzile se pot edita pe una sau mai multe linii;
  - Clauzele sunt uzual plasate pe linii separate, dar nu obligatoriu;
  - Cuvintele predefinite nu pot fi separate pe mai multe linii;
  - Comenzile nu sunt *case sensitive* (numai datele stocate in baza de date).

Operatori SQL

- **Operatori de comparatie**
  Sunt operatori folositi pentru compararea valorilor coloanelor intre ele sau cu valori numerice si pot fi de doua feluri:
- ✓ **Operatori logici**

Operator	Semnificatie
=	egal cu
>	mai mare decit
>=	mai mare sau egal
<	mai mic decit
<=	mai mic sau egal

Operatori SQL

- ✓ **Operatori SQL**

Operator	Semnificatie
BETWEEN..AND...	intre doua valori(inclusiv)
IN(list)	compara cu o lista de valori
LIKE	compara cu un model de tip caracter
IS NULL	este o valoare nula

Operatori SQL

- **Operatorii de negatie**

  Sunt operatori folositi pentru negarea valorilor coloanelor sau verificarea conditiilor de inegalitate si pot fi de doua feluri:

- ✓ **Operatori logici**

Operator	Semnificatie
!=	diferit de (VAX,UNIX,PC)
^=	diferit de (IBM)
<>	diferit de (toate OS)
NOT col_name =	diferit de
NOT col_name >	mai mic sau egal

Operatori SQL

- ✓ **Operatori SQL**

Operator	Semnificatie
NOT BETWEEN	nu se afla intre doua valori date
NOT IN	nu se afla intr-o lista data de valori
NOT LIKE	diferit de un sir
IS NOT NULL	nu este o valoare nula

- Folosind operatorul LIKE asociat cu simbolurile urmatoare, este posibil sa selectam randurile care se potrivesc cu un sir sau subsir de caractere :

Simbol	Semnificatie
%	orice secventa de mai multe caractere
_	un singur caracter

Operatori SQL

Observatii:

- Daca se compara o coloana sau expresie cu NULL, atunci operatorul de comparatie trebuie sa fie IS sau IS NOT. Daca se foloseste orice alt operator rezultatul va fi totdeauna FALSE (de exemplu expresia comision != NULL este intotdeauna falsa).
- Operatorii AND si OR pot fi utilizati pentru a compune expresii logice cu conditii multiple. Predicatul AND este adevarat numai daca ambele conditii sunt TRUE, iar predicatul OR este adevarat daca cel putin una dintre conditii este TRUE. Se pot combina AND sau OR in acceasi expresie logica in clauza WHERE, iar in acest caz operatorii AND sunt evaluati primii si apoi operatorii OR (deci operatorul AND au o precedenta mai mare decat OR).

Operatori SQL

- Daca operatorii au precedenta egala, atunci ei se evalueaza de la stanga la dreapta.
- Precedenta operatorilor logici este urmatoarea:

1. Operatorii de comparatie si operatorii SQL au precedenta egala:

   =, >= , <>, BETWEEN...AND, IN, LIKE, IS NULL.

2. NOT - cand este folosit pentru a inversa rezultatul unei expresii logice (de exemplu SELECT ….WHERE not(sal>2000))
3. AND
4. OR.

- Pentru a fi siguri de ordinea de executie a doua operatii, se recomanda folosirea parantezelor rotunde pentru gruparea operatiilor.

Crearea unei tabele

➢ **Comanda CREATE TABLE**

- Este folosita pentru crearea unei tabele in baza de date. Aceasta comanda va fi prezentata in detaliu intr-un alt capitol.

Exemplu:

- Vom crea doua tabele pentru evidenta departamentelor si a angajatilor:

```
CREATE TABLE departamente
( id_dep        number(2)    not null,
  den_dep       varchar2(30),
  telefon       varchar2(10)  );

CREATE TABLE angajati
( id_ang        number(4)     not null,
  nume          varchar2(30),
  functie       varchar2(20),
  id_sef        number(4),
  data_ang      date,
  salariu       number(7,2),
  comision      number(7,2),
  id_dep        number(2) );
```

Inserarea datelor intr-o tabela

➢ **Comanda INSERT**

- Este folosita pentru inserarea datelor intr-o tabela si are urmatoarea sintaxa:

  INSERT INTO [schema.] table_name[view_name][@dblink]
  [column 1, column 2, ...]
  VALUES (expr1, expr2. ....) subquery

  unde :

- **schema** – este schema unde este creata tabela (specifica userul si baza de date);
- **table_name** – este numele tabelei;
- **view_name** – este numele unui view creat pe o tabela;
- **column** – este numele coloanei;
- **expr** – reprezinta valoarea aferenta coloanei;
- **subquery** – este o subcerere care returneaza linii cu date din una sau mai multe tabele.

Inserarea datelor intr-o tabela

Exemple:

- Sa facem o inserare completa (toate coloanele au valori nenule) in tabelele create anterior. In acest caz nu trebuie sa specificam numele coloanelor dar valorile trebuie specificate in clauza VALUES in ordinea de creare a coloanelor in tabela.

SQL> INSERT INTO departamente VALUES (50, 'Proiectare Software', '0213262031');

SQL> INSERT INTO angajati VALUES (111, 'Popa Daniela', 'Inginer', 777, '1-OCT-2019', 750, 100, 10);

- Daca facem inserari numai in anumite coloane comanda arata astfel:

SQL> INSERT INTO angajati (id_ang,nume,functie) VALUES (777, 'Petrescu Florin', 'Contabil');

- Trebuie mentionat ca in acest caz trebuie specificate toate coloanele care fac parte dintr-o cheie primara sau sunt declarate NOT NULL la crearea tabelei, altfel se va genera un cod de eroare.

Inserarea datelor intr-o tabela

- Urmatoarea comanda va genera o eroare deoarece nu se specifica valoare pentru coloana id_ang care este declarata not null:

SQL> INSERT INTO angajati (nume, functie, salariu) VALUES (’Tache Marius’, ’Tehnician’, 1300);

ERROR at line 1:

ORA-01400: cannot insert NULL into ("SCOTT"."ANGAJATI"."ID_ANG")

- Nu se accepta inregistrari cu valori care depasesc dimensiunea coloanei. Urmatoarea comanda va genera o eroare deoarece se specifica o valoare prea mare pentru id_dep:

SQL> INSERT INTO departamente VALUES (222, ’Contabilitate’, '0213262032’);

*****

ERROR at line 1:

ORA-01438: value larger than specified precision allowed for this column

Stergerea datelor dintr-o tabela

➢ **Comanda DELETE**

- Este folosita pentru stergerea liniilor dintr-o tabela sau view si are urmatoarea sintaxa:

DELETE FROM [schema.]
table_name[view_name][@dblink]
WHERE condition
(column1,column2, ..) IN [NOT IN] subquery

unde :

- **schema** – este schema unde este creata tabela (specifica userul si baza de date);
- **table_name** – este numele tabelei;
- **view_name** – este numele unui view creat pe o tabela;
- **condition** – conditia care trebuie indeplinita pentru liniile sterse;
- **column** – este numele coloanei;
- **subquery** – este o subcerere care returneaza linii cu date din una sau mai multe tabele.

Stergerea datelor dintr-o tabela

Exemple:

- Stergerea unui angajat din tabela angajati se face cu comanda:

SQL> DELETE FROM angajati WHERE nume=’POPA DANIELA’ ;

- Stergerea tuturor angajatilor care au venit in companie inainte de anul 1981 se face cu comanda:
- **SQL> DELETE FROM angajati WHERE data_ang < ’1-JAN-1981’;**
- Pentru a sterge toti angajatii care s-au angajat in luna Noiembrie, indiferent de an, folosim comanda:

SQL> DELETE FROM angajati WHERE data_ang LIKE '%NOV%’;

- Stergerea angajatilor care nu au sef (id_sef este null) se face astfel:

SQL> DELETE FROM angajati WHERE id_sef is null;

- Pentru a sterge toate liniile din tabela angajati folosim comanda:

SQL> DELETE FROM angajati;

- Refacerea datelor sterse accidental se face cu comanda ROLLBACK:

SQL> ROLLBACK;

Modificarea datelor dintr-o tabela

- **Comanda UPDATE**

Este folosita pentru modificarea datelor in baza de date si are urmatoarea sintaxa:

UPDATE [schema.] table_name[view_name][@dblink]
SET column=expr
column=subquery
(column1,column2,...) IN [NOT IN] subquery
WHERE condition

unde :

- **schema** – este schema unde este creata tabela (specifica userul si baza de date);
- **table_name** – este numele tabelei;
- **column** – este numele coloanei;
- **condition** – conditia pentru modificarea liniilor;
- **subquery** – este o subcerere care returneaza linii cu date din una sau mai multe tabele.

Modificarea datelor dintr-o tabela

Exemple:

- Modificarea salariului si comisionului unui angajat se face astfel:

SQL> UPDATE angajati SET salariu=1200, comision=100 WHERE nume='IONESCU VICTOR';

SQL> UPDATE angajati SET salariu=1000 WHERE id_ang=7369;

- Modificarea tuturor salariilor prin indexare cu 10%:

SQL> UPDATE angajati SET salariu=salariu*1.1;

- Daca dorim sa crestem salariile doar pentru angajatii din departamentul 10 folosim comanda:

SQL> UPDATE angajati SET salariu=salariu*1.1 WHERE id_dep=10;

- Acordarea unui comision pentru angajatii veniti in companie in anul 1981 se face astfel:

SQL> UPDATE angajati SET comision=0.1*salariu WHERE data_ang>'1-JAN-1981' AND data_ang<'31-DEC-1981';

- In absenta clauzei WHERE toate liniile vor fi actualizate.

Modificarea datelor dintr-o tabela

- Cand se face actualizarea datelor intr-o tabela se verifica automat si constrangerile de integritate definite pe tabela respectiva, altfel comanda genereaza un cod de eroare si tranzactia esueaza.
- Situatiile in care pot sa apara erori sunt:
  - Noile valori fac duplicare de cheie primara sau unica;
  - Actualizarea valorii cu o valoare nula cand coloana este NOT NULL;
  - Valorile noi nu respecta o constrangere CHECK;
  - Valorile noi nu respecta o constrangere FOREIGN KEY;
  - Valorile vechi erau referite de alte tabele printr-o constrangere FOREIGN KEY;

Vizualizarea datelor dintr-o tabela

➢ **Comanda SELECT**

- Este folosita pentru vizualizarea datelor dintr-o tabela. Aceasta comanda va fi prezentata in detaliu intr-un alt capitol.

Exemple:

SQL> SELECT * FROM angajati;

SQL> SELECT nume, functie, salariu FROM angajati;

SQL> SELECT nume, functie, salariu, comision FROM angajati WHERE id_dep=10;

SQL> SELECT * FROM angajati WHERE functie='DIRECTOR';

- Datele din baza de date sunt case sensitive. Urmatoarea comanda nu va returna niciun rand, cu toate ca exista angajati cu functia DIRECTOR:

SQL> SELECT * FROM angajati WHERE functie='Director';

no rows selected

Cereri de interogare SQL

Alexandru Boicea Curs: Baze de date

Cereri de interogare SQL*Plus

- Cererile de interogare SQL folosesc in exclusivitate comanda **SELECT**, fiind utilizate atat pentru interogarea obiectelor create de utilizatori cat si a celor sistem. Sintaxa comenzii este urmatoarea :

  SELECT [DISTINCT,ALL] [schema.table.]expresion expr_alias

  FROM [schema.table@dblink] table_alias

  [WHERE condition]

  [START WITH condition][CONNECT BY condition]

  [UNION,UNION ALL,INTERSECT,MINUS][SELECT command]

  [GROUP BY expresion][HAVING condition]

  [ORDER BY expresion(position)] [ASC,DESC]

  [FOR UPDATE OF schema.table.column] [NOWAIT]

Cereri de interogare SQL*Plus

- Parametrii comenzii au urmatoarea semnificatie(cei din paranteze sunt optionali):
- ***DISTINCT*** - returneaza o singura linie in cazul in care cererea returneaza linii duplicate;
- ***ALL*** – returneaza toate liniile simple si duplicate;
- ***schema.table*** – reprezinta shema de identificare a tabelei(sau view-lui) specificata prin *user.table_name;*
- ***expresion*** – reprezinta un nume de coloana sau o expresie care poate folosi functii sistem (* selecteaza toate coloanele tabelelor din clauza FROM);
- ***expr_alias*** – este un nume alocat unei expresii care va fi folosit in formatarea coloanei (apare in antetul listei);
- ***dblink*** – reprezinta numele complet sau partial de identificare a unei baze de date (database.domain@connection_qualifier)

Cereri de interogare SQL*Plus

- ***table_alias*** – este un nume alocat unei tabele(view) care va fi folosit in cereri corelate;
- ***WHERE condition*** – reprezinta o clauza (inlantuire de conditii) care trebuie sa fie indeplinita in criteriul de selectie a liniilor;
- ***START WITH condition*** – stabileste criteriul de selectie pentru prima linie;
- ***CONNECT BY condition*** – stabileste o ierarhie de selectie a liniilor;
- ***GROUP BY expresion*** – stabileste criteriile de grupare a liniilor(numele coloanelor folosite in criteriul de grupare);
- ***HAVING condition*** – restrictioneaza liniile din grup la anumite conditii;

Cereri de interogare SQL*Plus

- ***UNION,UNION ALL,INTERSECT,MINUS*** – face operatii pe multimi de linii selectate de mai multe comenzi *SELECT* prin aplicarea anumitor restrictii;
- ***ORDER BY expresion(position)*** – ordoneaza liniile selectate dupa coloanele din expresie sau in ordinea coloanelor specificate prin pozitie;
- ***FOR UPDATE OF*** – face o blocare (*lock*) a liniilor in vederea modificarii anumitor coloane;
- ***NOWAIT*** – returneaza controlul userului daca comanda asteapta eliberarea unei linii blocata de un alt user.

Cereri simple de interogare

1. Cereri simple

- Cererea urmatoare returneaza toate coloanele si toate inregistarile continute de o tabela:

```
SQL> SELECT * FROM angajati;
```

- Interogari pe anumite coloane ale unei tabele:

```
SQL> SELECT id_dep, den_dep
       FROM departamente;
```

- Interogari care returneaza calcule pe anumite coloane si asigneaza un alias:

```
SQL> SELECT id_ang ecuson, nume,
       salariu*12+nvl(comision,0) venit_anual
       FROM angajati;
```

Cereri simple de interogare

- Interogari care concateneaza anumite coloane:

```
SQL> SELECT id_ang||'-'||nume angajat ,  functie, data_ang
       FROM angajati;
```

- Interogari care adauga coloane noi in lista:

```
SQL> SELECT id_ang||'-'||nume angajat ,  functie,
      salariu*12+nvl(comision,0) venit_lunar,  '        ' semnatura
      FROM angajati;
```

Cereri cu clauza WHERE

2. Cereri cu clauza WHERE

- Clauza WHERE poate compara valori de coloana ,valori literale, expresii aritmetice (sau functii) si poate avea patru tipuri de parametri:
    - nume de coloana;
    - operator de comparatie;
    - operator de negatie;
    - lista de valori.

Cereri cu clauza WHERE

- Lista persoanelor angajate in anul 1980:

```
SQL> SELECT id_ang ecuson,  nume, functie, data_ang
     FROM angajati
     WHERE data_ang LIKE '%80';
```

- Lista persoanelor al caror nume incepe cu liteara ***F*** si au numele functiei pe 7 caractere :

```
SQL> SELECT id_ang ecuson,  nume, functie, data_ang
     FROM angajati
     WHERE nume LIKE 'F%' and functie LIKE '_______';
```

Cereri cu clauza WHERE

- Lista angajatilor din departamentul 20 care nu au primit comision:

```
SQL> SELECT id_ang ecuson,  nume, functie, salariu  FROM angajati
     WHERE  (comision=0 OR comision IS NULL) AND id_dep=20
     ORDER BY nume;
```

- Lista angajatilor care au primit comision si nu sunt directori:

```
SQL> SELECT id_ang ecuson,  nume, functie, salariu,comision
     FROM angajati
     WHERE  comision IS NOT NULL and  functie NOT LIKE  'DIRECTOR'
     ORDER BY nume;
```

Cereri cu variabile substituite

3. Cereri cu variabile substituite

- Cererile SQL pot fi executate folosind anumiti parametri, care se mai numesc si variabile substituite(sau de substitutie).

➢ **Variabile ampersand (&)**

O astfel de variabila se defineste sub forma **&nume_variabila** si este un parametru care se va introduce de la tastatura in timpul executiei comenzii in care este utilizata. Parametrul cu un singur ampersand trebuie introdus de fiecare data, chiar daca este folosit de mai ori in aceeasi comanda SQL.

Exemplu:

```
SQL> SELECT id_ang,nume,functie,salariu FROM angajati
        WHERE  id_sef=&ecuson_sef;
```

Enter value for ecuson_sef: 7698

- old 1: SELECT id_ang,nume,functie,salariu FROM angajati WHERE id_sef=&ecuson_sef
- new 1: SELECT id_ang,nume,functie, salariu FROM angajati WHERE id_sef=7698;

Cereri cu variabile substituite

➢ **Variabile dublu ampersand (&&)**

- Spre deosebire de variabila cu un singur ampersand, o variabila cu dublu ampersand va fi stocata si va putea fi apelata pe toata sesiunea de lucru.
- Definirea se face similar **&&nume_variabila** si va fi ceruta o singura data, folosirea ei de mai multe ori in cadrul comenzii se face apeland-o cu **&nume_variabila**.

Exemplu:

```
SQL>  SELECT nume,functie,&&venit  venit_lunar
          FROM  angajati WHERE &venit>2000;
Enter value for venit: salariu+nvl(comision,0)
```

- Pentru a reseta o variabila cu dublu ampersand se utilizeaza comanda UNDEFINE :

```
SQL>UNDEFINE  venit
```

Cereri cu variabile substituite

- **Variabile de sistem (&n)**

- Sunt variabile definite numeric (1-9) care sunt predefinite de catre sistem si care functioneaza similar cu variabilele cu dublu ampersand. Avantajul folosirii acestor variabile este ca pot fi apelate direct dintr-un fisier de comenzi indirecte, fara a fi definite in prealabil. Suporta valori numerice, alfanumerice si data calendaristica.

Exemplu:

```
SQL>  SELECT id_ang,nume,functie,data_ang FROM angajati
        WHERE functie='&1' and  data_ang>'&2'
        ORDER BY data_ang;
SQL> SAVE angajari.sql
```

Created file angajari.sql

SQL> START angajari PROGRAMATOR 15-AUG-1981

Cereri definite cu ACCEPT

➢ **Variabile definite cu ACCEPT**

- Cand definim o variabila cu ampersand, totdeauna promtul va fi numele variabilei.
- Folosind comanda ACCEPT, se poate redefini promtul si chiar se pot ascunde caracterele introduse de la tastatura(facilitate utila in cazul unei parole).

Exemplu:

Se vor edita urmatoarele comenzi in fisierul **c:\temp\functia.sql**

```
SQL> ACCEPT functie_sef CHAR
        PROMPT 'Introduceti functia  sefului:' ;
SQL> SELECT nume,salariu,comision FROM angajati
        WHERE functie='&functie_sef';
```

Fisierul se va rula in SQL*Plus si se va introduce functia sefului:

```
SQL>  @c:\temp\functia.sql
```

Introduceti functia sefului: DIRECTOR

Cereri definite cu DEFINE

- **Variabile definite cu DEFINE si resetate cu UNDEFINE**

- Variabilele definite cu DEFINE nu vor mai afisa promptul atunci cand sunt setate si raman setate pana cand vor fi resetate cu comanda UNDEFINE.

Exemple:

```
SQL> DEFINE procent_prima= 1.15
SQL> SELECT nume, salariu, salariu*&procent_prima prima
        FROM  angajati WHERE id_dep=20;

SQL> DEFINE venit= salariu+nvl(comision,0)
SQL> SELECT nume, data_ang, &venit  venit_lunar FROM angajati
        WHERE functie='DIRECTOR';
SQL> UNDEFINE procent
SQL> UNDEFINE venit
```

Cereri de interogare SQL*Plus

Exercitii:

1. Sa se faca o lista cu toti angajatii care s-au angajat inainte de anul 1982 si nu au primit comision.

a)

```
SQL> SELECT * FROM angajati
        WHERE data_ang<'1-JAN-1982'  and
                (comision is null or comision =0);
```

b)

```
SQL> SELECT * FROM angajati
        WHERE data_ang<'1-JAN-1982'  and   nvl(comision,0)=0;
```

Cereri de interogare SQL*Plus

2. Sa se faca o lista cu toti angajatii care au salariul peste 3000 $ si nu au sefi, ordonati dupa departamente si nume.

```
SQL> SELECT * FROM angajati
        WHERE salariu>3000  and id_sef is null
        ORDER BY id_dep,nume;
```

3. Sa se faca o lista cu numele, functia si venitul anual al angajatilor care nu sunt directori pentru un departament introdus de la tastatura.

a)

```
SQL> SELECT nume, functie, salariu*12+nvl(comision,0) venit_anual
        FROM angajati
        WHERE functie not like 'DIRECTOR' and   id_dep=&nr_depart;
```

Cereri de interogare SQL*Plus

b)

```
SQL> DEFINE venit_anual='salariu*12+nvl(comision,0)';
SQL> SELECT nume,functie,&venit_anual  FROM angajati
        WHERE functie not like 'DIRECTOR' and id_dep=&nr_depart;
SQL> UNDEFINE venit_anual ;
```

4. Sa se faca o lista cu departamentul, numele, data angajarii si salariul tuturor persoanelor angajate in anul 1981, din doua departamente pentru care id_dep se introduce de la tastatura .

a)

```
SQL> SELECT id_dep,nume,data_ang,salariu  FROM angajati
        WHERE data_ang like '%81' and
                (id_dep=&nr_depart1 or id_dep=&nr_depart2)
        ORDER BY  id_dep;
```

Cereri de interogare SQL*Plus

5. Sa se scrie o comanda SQL care listeaza toti angajati dintr-un departament (introdus ca parametru de la tastura), care au venitul anual peste un venit mediu anual (introdus tot de la tastatura).

a)

```
SQL> SELECT  id_dep,id_ang,nume||'-'||functie
                    nume_functie,salariu, comision,
                    salariu*12+nvl(comision,0) venit_anual
       FROM angajati
       WHERE id_dep=&departament and
                 (salariu*12+nvl(comision,0))> &venit;
```

Cereri de interogare SQL*Plus

b)

```
SQL> DEFINE  venit_anual=’(salariu*12+nvl(comision,0))’;
SQL> SELECT  id_dep,id_ang,nume||'-'||functie
                        nume_functie,salariu, comision,
                        salariu*12+nvl(comision,0) venit_anual
        FROM angajati
        WHERE id_dep=&departament  and  &venit_anual> &venit;
SQL> UNDEFINE venit_anual ;
```

Cereri de interogare SQL*Plus

c)

```
SQL> DEFINE venit_anual= ’(salariu*12+nvl(comision,0))’;
SQL> ACCEPT depart number PROMPT  ’Departament:’;
SQL> ACCEPT val_venit number PROMPT  ’Venit anual:’;
SQL> SELECT  id_dep,id_ang,nume||'-'||functie
                        nume_functie,salariu, comision,
                        salariu*12+nvl(comision,0) venit_anual
        FROM angajati
        WHERE id_dep=&depart and  &venit_anual> &val_venit;
SQL> UNDEFINE venit_anual ;
SQL> UNDEFINE val_venit ;
SQL> UNDEFINE depart ;
```

Metode de JOIN

Alexandru Boicea Curs: Baze de date

Metode de JOIN

- Pentru extragerea datelor din mai multe tabele din baza de date, comanda SELECT foloseste una sau mai multe metode de JOIN. Sintaxa pentru un join simplu este urmatoarea:

SELECT [DISTINCT,ALL] [table].expresion expr_alias
FROM [schema.table1] table1_alias,
[schema.table2] table2_alias,
WHERE table1_alias.column=table2_alias.column
ORDER BY expresion(position)] [ASC,DESC]

Metode de JOIN

- Parametrii comenzii au urmatoarea semnificatie(cei din paranteze sunt optionali):
  - ***DISTINCT*** - returneaza o linie in cazul in care cererea returneaza linii duplicate;
  - ***ALL*** – returneaza toate liniile simple si duplicate;
  - ***schema.table*** – reprezinta shema de identificare a tabelei(sau view-lui) specificata prin *user.table_name;*
  - ***expresion*** – reprezinta un nume de coloana sau o expresie care poate folosi functii sistem (* selecteaza toate coloanele tabelelor din clauza FROM);
  - ***expr_alias*** – este un nume alocat unei expresii care va fi folosit in formatarea coloanei (apare in antetul listei);

Metode de JOIN

- ***table_alias*** – este un nume alocat unei tabele(sau view) care va fi folosit in cereri corelate;
- ***WHERE condition*** – reprezinta o clauza (inlantuire de conditii) care trebuie sa fie indeplinita in criteriul de selectie a liniilor;
- ***ORDER BY expresion(position)*** – ordoneaza liniile selectate dupa coloanele din expresie sau in ordinea coloanelor specificate prin pozitie.

Equi JOIN

➢ Equi-join

- Daca in conditia de join apar numai egalitati, avem de-a face cu un *equi-join*. Pentru a putea sa realizam un join pe mai multe tabele, este obligatoriu ca ele sa contina coloane de acelasi tip, cu date comune sau corelate.

Exemplu:

- Pentru a lista angajatii dintr-un departament folosim tabela *angajati,* insa in aceasta tabela gasim asociat fiecarui angajat un id_dep, iar denumirea departamentului respectiv o gasim in tabela *departamente.*
- Se impune un join intre cele doua tabele, pentru ca in lista sa apara si denumirea departamentului.

Equi JOIN

```
SQL>  SELECT a.id_dep ,b.den_dep depart,
      a.nume, a.functie
      FROM angajati a, departamente b
      WHERE   a.id_dep=b.id_dep and
      a.id_dep=10;
```

- Se observa ca au fost folosite aliasuri pentru tabele pentru a nu crea ambiguitate cand referim coloane cu aceeasi denumire.

Non Equi JOIN

➢ Non Equi-join

- Se foloseste cand in conditia de join avem alti operatori in afara de operatorul de egalitate sau cand doua sau mai multe tabele nu au coloane comune, dar trebuie totusi relationate.

Exemple:

- Relatia dintre tabelele *angajati* si *grila_salariu* este un non-equi-join, in care nicio coloana din prima tabela nu corespunde direct cu o coloana din cealalta.
- Relatia se obtine folosind tot un operator, altul decat = , de exemplu *between*.
- Pentru a evalua gradul de salarizare al unui angajat, trebuie sa consultam grila de salarizare pentru a identifica in ce plaja de salariu se incadreaza salariul:

Non Equi JOIN

```
SQL>  SELECT a.nume, a.salariu, b.grad
      FROM angajati a, grila_salariu b
      WHERE a.salariu  BETWEEN b.nivel_inf
      AND b.nivel_sup  AND  a.id_dep=20;
```

- Sunt situatii cand trebuie sa folosim si equi-join si non equi-join intr-o cerere . In exemplul anterior, daca dorim sa listam si departamentul, va trebui sa facem join intre trei tabele:

```
SQL>  SELECT c.den_dep,a.nume, a.salariu, b.grad
      FROM angajati a, grila_salariu b, departamente c
      WHERE a.salariu  BETWEEN b.nivel_inf AND
      b.nivel_sup  AND  a.id_dep=c.id_dep  AND
      a.id_dep=20;
```

Self JOIN

➢ **Joinul unei tabele cu ea insasi (*self join*)**

- Sunt situatii cand avem nevoie sa extragem date corelate din aceeasi tabela. De exemplu, daca dorim sa afisam care sunt sefii angajatilor trebuie sa extragem din tabela *angajati* si numele sefului (id_sef).

```
SQL> SELECT a.nume nume_ang,a.functie functie_ang,
     b.nume nume_sef,b.functie functie_sef
     FROM angajati a, angajati b
     WHERE a.id_sef=b.id_ang AND a.id_dep=10;
```

Cross JOIN

➢ **Produs cartezian**

- Produsul cartezian a doua tabele se obtine prin concatenarea fiecarei linii dintr-o tabela cu fiecare linie din cealalta, rezultand un numar de linii egal cu produsul numarului de linii din fiecare tabela. Aceasta situatie este mai putin practica si se intalneste, de regula, cand sunt puse gresit conditii in clauza WHERE.

Exemplu:

```
SQL>  SELECT nume ,functie ,den_dep
      FROM angajati , departamente
      WHERE functie='DIRECTOR';
```

Outer JOIN

- **Join extern (*outer join*)**

- Folosind equi-join putem selecta toate liniile care indeplinesc conditiile din clauza WHERE. Apar situatii cand cererea trebuie sa selecteze si linii care nu indeplinesc toate conditiile din clauza.

Exemple:

- Sa construim structura organizatorica a firmei selectand toate departamentele si angajatii care fac parte din fiecare departament. Exista, in tabela departamente, departamentul 40 care nu are niciun angajat si folosind equi-join acesta nu apare in lista. Pentru a depasi situatia se foloseste un *join extern* (+) asa cum se vede mai jos:

Outer JOIN

```
SQL> SELECT a.id_dep, a.den_dep, b.nume, b.functie
     FROM departamente a, angajati b
     WHERE a.id_dep=b.id_dep(+);
```

- In lista va apare si departamentul VANZARI care nu are niciun angajat. Totdeanuna semnul (+) se pune in dreptul tabelei deficitare ca informatii.
- Putem sa folosim si operatorul *BETWEEN* intr-un join extern. Daca vrem sa aflam care angajati raman in grila de salarizare prin dublarea salariilor, executam urmatoarea cerere:

Outer JOIN

```
SQL>  SELECT c.den_dep,a.nume, a.salariu, b.grad
        FROM angajati a, grila_salariu b, departamente c
        WHERE a.salariu*2  BETWEEN b.nivel_inf(+) AND
         b.nivel_sup(+) AND  a.id_dep=c.id_dep ;
```

- Daca dorim sa selectam toate departamentele care au primul caracter din id_dep din tabela departamente folosit printre id_dep din tabela angajati, se poate folosi si operatorul *LIKE* :

```
SQL>  SELECT a.id_dep,a.den_dep,b.nume,b.functie
        FROM departamente a, angajati b
        WHERE b.id_dep(+)  LIKE substr(a.id_dep,1,1)||'%';
```

Vertical JOIN

- **Join vertical (*vertical join*)**

- Join-ul vertical este folosit pentru prelucrarea liniilor returnate de mai multe cereri *SELECT* si foloseste operatorii ***UNION*** (reuniune), ***INTERSECT*** (intersectie), ***MINUS*** (diferenta). In acest caz, join-ul se face dupa coloane de acelasi tip, nu dupa randuri, de aceea se mai numeste si vertical.

Exemple:

- Daca dorim lista angajatilor din departamentele 10 si 30, se poate utiliza cererea urmatoare:

Vertical JOIN

```
SQL>  SELECT  id_dep,nume,functie,salariu
      FROM angajati   WHERE id_dep=10
      UNION
      SELECT   id_dep,nume,functie,salariu
      FROM angajati  WHERE id_dep=30;
```

- Trebuie retinut ca reuniunea se poate face pe coloane declarate de acelasi tip (number, varchar, date), chiar daca au semnificatii diferite. Sa construim o cerere care reuneste pe aceeasi coloana salariile angajatilor din departamentul 10 si cu comisioanele celor din departamentul 30.

Vertical JOIN

```
SQL>  SELECT id_dep,nume,functie,'are salariu'
       salar_comision, salariu sal_com
       FROM angajati WHERE id_dep=10
      UNION
      SELECT id_dep,nume,functie,'are comision ',
      comision FROM  angajati WHERE id_dep=30;
```

- Folosind operatorul *UNION ALL* se selecteaza si liniile duplicate:

```
SQL>  SELECT functie FROM angajati WHERE  id_dep=10
      UNION ALL
      SELECT functie FROM angajati WHERE  id_dep=20;
```

Vertical JOIN

- Operatorul *INTERSECT* este folosit pentru a selecta liniile comune. Daca dorim sa aflam care sunt functiile angajatilor care au primit acelasi comision si care se regasesc in toate departamentele, scriem urmatoarea cerere:

```
SQL>  SELECT functie, comision FROM angajati
      WHERE  id_dep=10
      INTERSECT
      SELECT functie,comision FROM angajati
      WHERE  id_dep=20
      INTERSECT
      SELECT functie,comision FROM angajati
       WHERE  id_dep=30;
```

Vertical JOIN

- Pentru a afla care sunt functiile din departamentul 10 care nu se regasesc in departamentul 30, folosim operatorul *MINUS* :

```
SQL> SELECT functie FROM angajati WHERE id_dep = 10
     MINUS
     SELECT functie FROM angajati WHERE id_dep = 30;
```

Reguli de JOIN

- Observatii:
  - Atunci cand conditia de join lipseste, fiecare linie a unei tabele din lista FROM este asociata cu fiecare linie a celorlalte tabele, obtinandu-se de fapt ***produsul cartezian*** al acestora.
  - Daca in conditia de join apar numai egalitati operatia este numita si ***equi-join***. In celelalte cazuri avem un ***non-equi-join***.
  - In lista de tabele care participa la join o tabela poate sa apara repetat. O astfel de operatie este numita si ***joinul unei tabele cu ea insasi*** (***self-join***).
  - In cazul in care o linie a unei tabele nu se coreleaza prin conditia de join cu nicio linie din celelalte tabele ea nu va participa la formarea rezultatului. Se poate insa obtine ca aceasta sa fie luata in considerare pentru rezultat, folosind un ***join extern*** (***outer join***).

Reguli de JOIN

- In cazul general al unui **join pe N tabele, conditia de join este compusa din N - 1 subconditii** conectate prin AND care relationeaza intreg ansamblul de tabele. Altfel spus, daca se construieste un graf al conditiei in care nodurile sunt tabele si arcele subconditii de join care leaga doua tabele, atunci acest graf trebuie sa fie conex.
- Marcajul de **join extern se poate folosi si atunci cand conditia de join este compusa**, cu exceptia cazului in care se foloseste OR sau operatorul de incluziune IN urmat de o lista care contine mai mult de o valoare.
- Joinul extern **se poate folosi si in conjunctie cu operatorii specifici SQL.**

Metode de JOIN in SQL-3

- Pana la versiunea Oracle 9i sintaxa joinului in Oracle era diferita de standardul ANSI (*American National Standards Institute*). Incepand cu aceasta versiune au fost introduse in limbaj si tipurile de join din standardul SQL-3(anul 1999) printre care ***cross-join, join natural*** si mai multe ***variante de join extern:***
  - **CROSS JOIN** – join pe produs cartezian
  - **[INNER] JOIN … USING** – join pe coloane comune
  - **[INNER] JOIN … ON** – join general
  - **NATURAL JOIN** – join natural
  - **OUTER JOIN … ON** – join extern
- In clauza FROM avem perechi de tabele care participa la join.

Cross JOIN in SQL-3

➢ CROSS JOIN

- Este folosit pentru obtinerea produsul cartezian si are urmatoarea sintaxa:

SELECT [DISTINCT|ALL]
[[table|table_alias].]{column|expression}
[column_alias]
FROM
[schema.]table1 [table1_alias] CROSS JOIN
[schema.]table2 [table2_alias]
[other clauses]

Cross JOIN in SQL-3

Exemple:

```
SQL>  SELECT a.nume, b.den_dep  FROM angajati a
      CROSS JOIN departamente b;
```

- Pentru a face JOIN si CROSS JOIN adaugam conditia de join in clauza WHERE:

```
SQL>  SELECT a.nume, a.data_ang, b.den_dep, b.sediu
      FROM angajati a
      CROSS JOIN departamente b
       WHERE a.id_dep=b.id_dep;
```

- Observatie: Conditia CROSS JOIN poate sa lipseasca in acest caz, rezultatul va fi acelasi.

Inner JOIN in SQL-3

- **JOIN … USING**

- Este un equi-join dupa coloane cu acelasi nume specificate in USING (dar nu toate). Sintaxa este urmatoarea:

SELECT [DISTINCT|ALL]
{{table|table_alias}.}{column|expression}
[column_alias]

FROM

[schema.]table1 [table1_alias]

[INNER] JOIN [schema.]table2 [table2_alias]

USING (column 1, column 2…)

[other clauses]

Inner JOIN in SQL-3

Exemple:

- Deoarece in ANGAJATI si DEPARTAMENTE avem o coloana cu acelasi nume (ID_DEP)) putem face un equi-join astfel:

```
SQL>  SELECT id_dep, den_dep, nume, data_ang, sediu
      FROM angajati
      INNER  JOIN departamente  USING (id_dep);
```

- Dupa cum se observa in lista USING numele coloanelor dupa care se efectueaza joinul nu trebuie prefixat cu numele sau aliasul vreuneia dintre tabele (coloanele comune se afiseaza o singura data).
- Daca se doreste selectia doar dintr-un departament se va adauga conditia suplimentara pe WHERE:

```
SQL>  SELECT id_dep, nume, den_dep,  data_ang, sediu
      FROM angajati
      JOIN departamente  USING (id_dep)
      WHERE den_dep='CONTABILITATE';
```

Inner JOIN in SQL-3

➢ **JOIN .. ON**

- Prin aceasta clauza se implementeaza un join general. Conditia de join (si eventual si conditiile suplimentare) se pun in clauza ON. Sintaxa este urmatoarea:

```
SELECT [DISTINCT|ALL]
{{table|table_alias}.}{column|expression} [column_alias]
FROM
[schema.]table1 [table1_alias]
[INNER] JOIN [schema.]table2 [table2_alias]
ON   {{table1|table1_alias}.}column_fromTable1 =
       {{table2|table2_alias}.}column_fromTable2
[other clauses]
```

Inner JOIN in SQL-3

Exemple:

- Cererea urmatoare efectueaza joinul dupa coloana ID_DEP si contine si o conditie suplimentara(filtru) dupa functia DIRECTOR:

```
SQL>  SELECT a.nume, a.data_ang, b.den_dep, b.telefon
      FROM angajati a JOIN departamente b
      ON (a.id_dep=b.id_dep AND  a.functie='DIRECTOR');
```

Conditia suplimentara se putea pune si in clauza WHERE.

- Intr-un JOIN ON se poate folosi si non-equi-join. Daca se doreste o lista a angajatilor cu salariul intre 1000 si 3000 se poate introduce urmatorul filtru:

```
SQL>  SELECT a.nume, a.data_ang, b.den_dep
      FROM angajati a  JOIN departamente b
      ON  (  a.id_dep=b.id_dep  AND
             a.salariu BETWEEN  1000  AND  3000);
```

Inner JOIN in SQL-3

- Observatie: Un join general se poate aplica pe mai multe tabele.

Exemplu:

- Cererea urmatoare efectueaza joinul tabelei de angajati cu tabelele de departamente si grila de salarii:

```
SQL>  SELECT b.den_dep , a.nume, a.salariu, c.grad
      FROM angajati a
      JOIN departamente b
        ON (a.id_dep=b.id_dep
            AND salariu+nvl(comision,0) >1000 )
      JOIN grila_salariu c
        ON (a.salariu>=c.nivel_inf  AND
            a.salariu<=c.nivel_sup)
      ORDER BY 1;
```

Natural JOIN in SQL-3

➢ **NATURAL JOIN**

- Este un equi-join dupa coloane cu acelasi nume si are sintaxa urmatoare:

SELECT [DISTINCT|ALL]
[[table|table_alias].]{column|expression}
[column_alias]
FROM [schema.]table1 [table1_alias]
NATURAL JOIN [schema.]table2 [table2_alias]
[other clauses]

Inner JOIN in SQL-3

Exemplu:

- Tabele DEPARTAMENTE si ANGAJATI au o coloana comuna(ID_DEP) dupa care se poate face un join natural:

```
SQL>  SELECT id_dep, nume, data_ang, den_dep, sediu
      FROM angajati
      NATURAL  JOIN  departamente ;
```

- Observatii:
  - In cazul folosirii clauzei NATURAL JOIN cele doua coloane trebuie sa aiba acelasi nume.
  - Daca coloanele au acelasi nume, nu se tine cont de tipul si semnificatia coloanelor.
  - Nu se accepta aliasuri pentru coloanele comune.

Outer JOIN in SQL-3

➢ **OUTER JOIN … ON**

- Join-ul extern se foloseste cand se doreste ca in rezultat sa apara si liniile cu valori nule pe coloanele corespondente din tabelele relationate. Sintaxa este urmatoarea:

```
SELECT [DISTINCT] lista_de_expresii
FROM tabela1
LEFT   \
RIGHT   | OUTER  JOIN tabela2
FULL   /
ON (tabela1.nume_coloana1 =
        tabela2.numecoloana2)
```

Left Outer JOIN in SQL-3

➢ **LEFT OUTER JOIN**

- In cazul join-ului extern stanga valorile nule provin din tabela2(cea deficitara ca date).

```
SELECT [DISTINCT|ALL]
{{table|table_alias}.}{column|expression}
[column_alias]
FROM
[schema.]table1 [table1_alias]
LEFT [OUTER ] JOIN [schema.]table2 [table2_alias]
ON   {{table1|table1_alias}.}column_fromTable1 =
       {{table2|table2_alias}.}column_fromTable2
[other clauses]
```

Left Outer JOIN in SQL-3

Exemplu:

- Cererea de mai jos face o lista cu toate departamente si angajatii lor, dar afiseaza si departamentele care nu au niciun angajat:

```
SQL>  SELECT a.nume nume_ang, a.data_ang, a.salariu, b.den_dep
                departament
      FROM departamente b
      LEFT OUTER JOIN angajati a ON(a.id_dep=b.id_dep);
```

- Urmatoarea cerere este echivalenta cu prima:

```
SQL>  SELECT a.nume nume_ang, a.data_ang, a.salariu, b.den_dep
                departament
      FROM angajati a , departamente b
      WHERE a.id_dep(+)=b.id_dep;
```

- Notatia pentru join extern stanga este: R $\triangleleft o \triangleright_L$ S. In rezultat apar toate liniile tabelei din stanga operatorului, inclusiv valorile nule.

Right Outer JOIN in SQL-3

➢ RIGHT OUTER JOIN

- In cazul join-ului extern dreapta valorile nule provin din tabela1(cea deficitara ca date). Sintaxa este similara cu join extern stanga.

Exemplu:

- Cererea urmatoare face o lista cu angajatii si sefii lor, inclusiv angajatii care nu au sefi:

```
SQL> SELECT b.id_ang sef, b.nume nume_sef, a.nume nume_ang,
                a.data_ang, a.salariu
     FROM angajati b
     RIGHT OUTER JOIN angajati a ON(a.id_sef= b.id_ang);
```

- Urmatoarea cerere este echivalenta cu prima:

```
SQL> SELECT b.id_ang sef, b.nume nume_sef, a.nume nume_ang,
                a.data_ang, a.salariu
     FROM angajati b, angajati a
     WHERE a.id_sef= b.id_ang(+);
```

- Notatia pentru join extern dreapta este: $R \triangleleft o \triangleright_R S$. In rezultat apar toate liniile tabelei din dreapta operatorului, inclusiv valorile nule.

Outer JOIN in SQL-3

- Observatie: Sa analizam urmatoarele cereri:

```
SQL>  SELECT  b.id_ang sef, b.nume nume_sef, a.nume nume_ang,
      a.data_ang, a.salariu
      FROM angajati a
      LEFT OUTER JOIN angajati b ON (a.id_sef = b.id_ang);

SQL>  SELECT  b.id_ang sef, b.nume nume_sef, a.nume nume_ang,
      a.data_ang, a.salariu
      FROM angajati b
      RIGHT OUTER JOIN angajati a ON (a.id_sef = b.id_ang);
```

- Daca executam cele doua cereri vom obtine acelasi rezultat.
- Practic, daca inversam tipul de join si aliasurile de coloana vom obtine doua cereri identice.

Full Outer JOIN in SQL-3

➢ **FULL OUTER JOIN**

- In cazul unui join extern complet rezultatul contine toate liniile din rezultatul joinului general si joinul extern LEFT/RIGTH, obtinut cu aceeasi conditie . Sintaxa este urmatoarea:

```
SELECT [DISTINCT|ALL]
{{table|table_alias}.}{column|expression} [column_alias]
 FROM
[schema.]table1 [table1_alias]
FULL [OUTER] JOIN [schema.]table2 [table2_alias]
ON   {{table1|table1_alias}.}column_fromTable1 =
       {{table2|table2_alias}.}column_fromTable2
[other clauses]
```

Full Outer JOIN in SQL-3

Exemplu:

- Cererea urmatoare returneaza o lista cu toti angajatii si toate departamentele:

```
SQL>  SELECT a.nume, a.data_ang, b.den_dep, b.sediu
      FROM angajati a
      FULL OUTER JOIN departamente b ON (a.id_dep=b.id_dep);
```

- Observatie: Urmatoarea cerere NU ESTE echivalenta cu prima:

```
SQL>  SELECT a.nume, a.data_ang, b.den_dep, b.sediu
      FROM angajati a,  departamente b
      WHERE a.id_dep(+)=b.id_dep(+);
ERROR at line 3: ORA-01468: a predicate may reference only one outer-joined table
```

- Notatia pentru join extern complet este: R ⟗ S. In rezultat apar toate liniile tabelelor din stanga si dreapta operatorului.

Full Outer JOIN in SQL-3

- Observatie: Un join extern se poate face pe mai multe tabele.

Exemplu:

- Cererea urmatoare returneaza o lista cu toate departamentele (inclusiv cele fara angajati), toti angajatii si toate gradele de salarizare(inclusiv pe cele care nu au corespondent in salariile angajatilor):

```
SQL> SELECT b.den_dep, a.nume, a.data_ang, c.grad
        FROM angajati a
        FULL OUTER JOIN departamente b
                        ON (a.id_dep=b.id_dep)
        FULL OUTER JOIN grila_salariu c
                        ON a.salariu between c.nivel_inf and c.nivel_sup
        ORDER BY b.den_dep;
```

Functii SQL

Alexandru Boicea Curs: Baze de date

Functii SQL

- Functia poate fi vazuta ca un operator de manipulare a datelor care accepta unul sau mai multe argumente (constanta , variabila , referire de coloana, etc.) si returneaza un rezultat.

  Sintaxa de apelare este urmatoarea:

  function_name(arg1, arg2,...)

- Rolul lor este de a face mai puternice cererile Oracle SQL*Plus si se pot folosi pentru:
  - Efectuarea calculelor numerice;
  - Prelucrare de siruri de caractere;
  - Prelucrarea datei calendaristice;
  - Schimbarea formatului pentru datele de afisare;
  - Conversia tipurilor de date.

Functii SQL

- Dupa specific si tipuri de argumente, functiile se pot imparti in urmatoarele categorii:
  - **Functii numerice**
  - **Functii pentru siruri**
  - **Functii pentru data calendaristica**
  - **Functii de conversie**
  - **Functii diverse** (accepta diverse tipuri de argumente)
  - **Functii de grup**

Functii numerice

➢ Functii numerice

- Aceste functii au ca parametri valori numerice si returneaza tot o valoare numerica.

Exemple:

- **ABS (n)** – returneaza valoarea absoluta a lui **n**

SQL> SELECT abs(-12) FROM dual; -- returneaza valoarea 12

- **CEIL (n)** – returneaza cel mai mic intreg >= **n**

SQL> SELECT ceil(14.7) FROM dual; -- returneaza valoarea 15

- **COS (n)** – returneaza **cosinus (n)** unde n este in radiani

SQL> SELECT cos(180 * 3.14/180) FROM dual; -- returneaza -1

- **COSH (n)** – returneaza cosinus hiperbolic

SQL> SELECT cosh(0) FROM dual; -- returneaza valoarea 1

Functii numerice

- **EXP (n)** – returneaza **e** la puterea **n (e**=2.7182..)

SQL> SELECT exp(4) FROM dual; -- returneaza valoarea 54.5981

- **FLOOR (n)** – returneaza cel mai mare intreg <= **n**

SQL> SELECT floor(11.6) FROM dual; -- returneaza valoarea 11

- **LN (n)** – returneaza logaritmul natural al lui **n** (n>0)

SQL> SELECT ln(95) FROM dual; -- returneaza valoarea 4.5538

- **LOG (m,n)** – returneaza logaritmul in baza **m** al lui **n**

SQL> SELECT log(10,100) FROM dual; -- returneaza valoarea 2

- **MOD (m,n)** – returneaza restul impartirii lui **m** la **n**

SQL> SELECT mod(14,5) FROM dual; -- returneaza valoarea 4

- **POWER (m,n)** – returneaza **m** la puterea **n**

SQL> SELECT power(3,2) FROM dual; -- returneaza valoarea 9

Functii numerice

- **ROUND (n[,m])** – returneaza **n** rotunjit la :

  m zecimale daca m>0;

  0 zecimale daca m este omis;

  m cifre inainte de virgula daca m<0;

SQL> SELECT round(15.193, 1) FROM dual; -- returneaza 15.2

SQL> SELECT round(15.193) FROM dual; -- returneaza 15

SQL> SELECT round(15.193, -1) FROM dual; -- returneaza 20

- **SIGN (n)** – returneaza -1 daca **n**<0, 0 daca **n**=0 si 1 daca **n**>0

SQL> SELECT sign(-17.5) FROM dual; -- returneaza valoarea -1

- **SIN (n)** – returneaza **sinus (n)** unde n este in radiani

SQL> SELECT sin(30 * 3.14/180) FROM dual; -- returneaza 0.5

- **SINH (n)** – returneaza cosinus hiperbolic

SQL> SELECT sinh(1) FROM dual; -- returneaza valoarea 1.1752

Functii numerice

- **SQRT (n)** – returneaza radacina patrata a lui **n** unde n>0

SQL> SELECT sqrt(26) FROM dual; -- returneaza valoarea 5.099

- **TAN (n)** – returneaza tangenta lui **n ,** unde n este in radiani

SQL> SELECT tan(135 * 3.14/180) FROM dual; -- returneaza -1

- **TAN (n)** – returneaza tangenta hiberbolica a lui **n ,**unde n este in radiani

SQL> SELECT tanh(0.5) FROM dual; -- returneaza 0.4621

- **TRUNC (n[,m])** – returneaza **n** trunchiat la :

  m zecimale daca m>0;

  0 zecimale daca m este omis;

  m cifre inainte de virgula daca m<0.

SQL> SELECT trunc(15.193, 1) FROM dual; -- returneaza 15.1

SQL> SELECT trunc(15.193) FROM dual; -- returneaza 15

SQL> SELECT trunc(15.193, -1) FROM dual; -- returneaza 10

Functii alfanumerice

- **Functii pentru siruri**

- Aceste functii accepta la intrare valori alfanumerice si returneaza o valoare numerica sau tot o valoare alfanumerica. Cele care returneaza valori de tip VARCHAR2 pot avea lungimea maxima 4000 caractere iar cele de tip CHAR pot avea lungimea maxima 2000 caractere.

Exemple:

- **CHR (n)** -returneaza caracterul care are reprezentarea binara **n**

SQL> SELECT chr(75) FROM dual; -- returneaza valoarea K

- **CONCAT (char1,char2)** – returneaza concatenarea lui **char1** cu **char2**

```
SQL> SELECT concat(concat(nume,' este '),functie) nume_functie
        FROM angajati WHERE id_ang=7839;
```

Functii alfanumerice

- **INITCAP (char)** – returneaza **char** cu majuscule

SQL> SELECT initcap('popescu mihai') FROM dual; -- returneaza **Popescu Mihai**

- **REPLACE (string,string1,[string2])** – inlocuieste in **string** caracterele din **string1** cu caracterele din **string2**

SQL> SELECT replace('JACK si JUE','J','BL') FROM dual;
-- returneaza valoarea **BLACK si BLUE**

- **TRANSLATE (col/string,string1,string2)** – translateaza in **col/string** caracterele specificate in **string1** in caracterele specificate in **string2**

SQL> SELECT nume,translate(nume,'C','P') FROM angajati WHERE id_dep=10; -- translateaza in nume litera **C** in **P**

- **RPAD/LPAD (char1,n,[char2])** – adauga la dreapta/stanga lui **char1** caracterele **char2** pana la lungimea **n**

SQL> SELECT rpad(nume,20,'*') completare_nume FROM angajati WHERE id_dep=10;

Functii alfanumerice

- **RTRIM (char[,set])** sterge din **char** ultimele caractere daca sunt in **set**

SQL> SELECT rtrim('Popescu','scu') FROM dual;--returneaza **Pope**

- **SUBSTR (char,m[,n])** – returneaza **n** caractere din **char** incepand cu pozitia **m**

SQL> SELECT substr ('Popescu ',2,3) FROM dual; -- returneaza **ope**

- **INSTR (char1,char2[,n[,m]])** – returneaza pozitia lui **cha**r2 incepand cu pozitia **n** la a **m**-a aparitie

SQL> SELECT instr('Protopopescu','op',3,2) FROM dual;
-- returneaza **7**

- **LENGTH (char)** – returneaza lungimea lui **cha**r ca numar de caractere

SQL> SELECT length('analyst') FROM dual; --returneaza **7**

- **SIGN(col/expr/value)**-returneaza **-1** daca col/exp/valoarea este un numar negativ , **0** daca este zero si **+1** daca este numar pozitiv

Functii pentru data calendaristica

➢ **Functii pentru data calendaristica**

- O baza de date ORACLE stocheaza datele calendaristice in urmatorul format intern:

  Secol / An / Luna / Zi / Ora / Minut / Secunda

- Formatul implicit de afisare sau intrare pentru o data calendaristica este *DD-MON-YY*.
- Plaja datei calendaristice este intre ’1-JAN-4712’ i.e.n si ’31-DEC-4712’ e.n.
- Toate functiile de tip data calendaristica intorc o valoare de tip *DATE* cu exceptia lui *MONTHS_BETWEEN* care intoarce o valoare numerica.

Functii pentru data calendaristica

Exemple:

- **ADD_MONTHS (date,n)** – returneaza o data prin adaugarea a **n** luni la **date**

```
SQL> SELECT nume, data_ang,add_months(data_ang,3)
        data_mod FROM angajati WHERE id_dep=10;
```

- **LAST_DAY (date)** – returneaza data ultimei zile din luna cuprinsa in **date**

```
SQL> SELECT nume,data_ang,last_day(data_ang) ultima_zi
        FROM angajati WHERE id_dep=10;
```

- **MONTHS_BETWEEN(date1,date2)** – returneaza numarul de luni (si fractiuni de luna) cuprinse intre **date1** si **date2**. Daca **date1**>**date2** rezultatul va fi pozitiv altfel va fi negativ.

```
SQL> SELECT nume, data_ang, sysdate,
        months_between(sysdate,data_ang) luni_vechime
        FROM angajati WHERE id_dep=10;
```

Functii pentru data calendaristica

- **NEXT_DAY(date,char)** – returneaza data urmatoarei zile **char** dupa **date**

```
SQL> SELECT next_day('17-NOV-2006', 'TUESDAY')
     marti_urmatoare  FROM dual;
```

- **ROUND(date,fmt)** – returneaza data prin rotunjirea lui **date** la formatul **fmt**

```
SQL> SELECT round(to_date('17-NOV-2006'), 'YEAR') rot_an
        FROM dual;
SQL> SELECT round(to_date('17-NOV-2006'), 'MONTH')
        rot_luna  FROM dual;
```

- **TRUNC(date,fmt)** – returneaza data prin trunchierea lui **date** la formatul **fmt**

```
SQL> SELECT trunc(to_date('17-NOV-2006'), 'YEAR') trunc_an
        FROM dual;
```

Functii pentru data calendaristica

- Folosind operatorii aritmetici **+** si **–** se pot face diferite operatii cu date calendaristice:
  - **data + numar** - aduna un numar de zile la data, returnand tot o data calendaristica;
  - **data - numar** - scade un numar de zile la data, returnand tot o data calendaristica;
  - **data1 – data2** - scade data2 din data 1, obtinand un nr. de zile;
  - **data + numar/24** - aduna la data un numar de ore pentru a obtine tot o data calendaristica.

Exemplu:

```
SQL> SELECT  data_ang,data_ang+7,data_ang-7,
        sysdate-data_ang FROM    angajati
        WHERE   data_ang LIKE '%JUN%';
```

Functii de conversie

➢ **Functii de conversie**

- Aceste functii fac conversia unui tip de data in alt tip de data.
- **TO_CHAR(date[,fmt[,'nlsparams']])** – face conversia unei date de tip DATE in format VARCHAR2

```
SQL> SELECT nume, to_char(data_ang, 'Month DD, YYYY')
        data_ang   FROM angajati
        WHERE to_char(data_ang,'YYYY') LIKE '1987';
```

- **TO_DATE(char[,fmt[,'nlsparams']])** – face conversia unei variabile de tip CHAR sau VARCHAR2 in format DATE

```
SQL> SELECT  to_date('15112006' ,'DD-MM-YYYY') data
        FROM dual;
```

Functii de conversie

- **TO_NUMBER(char[,fmt[,'nlsparams']])** – face conversia unei variabile de tip CHAR sau VARCHAR2 in format NUMBER

Exemplu:

- Pentru calculul primei in functie de anul angajarii, folosim cererea:

```
SQL> SELECT  nume, functie, salariu,
        salariu+to_number(to_char(data_ang,'yyyy'))/10 prima
        FROM angajati
        WHERE to_number(to_char(data_ang ,'YYYY'))=1987;
```

Functii de conversie

- Exemple de format pentru date numerice:

Format	Semnificatie	Exemple	
9	numere(nr.de 9 determina lungimea de afisare)	999999	1234
0	afiseaza zerourile de la inceput	099999	001234
$	simbolul dolar	$999999	$1234
.	punct zecimal	999999.99	1234.00
,	virgula	999,999	1,234
MI	semnele minus la dreapta(valori negative)	999999MI	1234-
PR	paranteze pentru numere negative	999999PR	<112>
EEEE	notatie stiintifica	99.999EEEE	1.234E+03
V	inmultire cu 10**n (n=numar de 9 dupa V)	9999V99	123400
B	afiseaza valori zero ca blancuri(nu 0)	B9999.99	1234.00

- Formatul de afisare a datelor se poate seta cu comanda COLUMN:

SQL>COLUMN nume FORMAT a30 -- format alfanumeric max 30 car

SQL>COL salariu FOR 99999.99 -- format numeric cu 2 zecimale

Functii de conversie

- **EXTRACT ([YEAR], [MONTH], [DAY], [HOUR], [MINUTE], [SECONDE] from datetime)** – extrage anul, luna, ziua, minutul, secunda din data calendaristica **datetime**.

```
SQL> SELECT   nume,  data_ang,
              extract (year from data_ang) ANUL,
              extract(month from data_ang) LUNA,
              extract (day from data_ang) ZIUA
     FROM angajati
     WHERE  functie =’ANALIST’ ;
```

Functii diverse

➢Functii diverse

- Aceste functii accepta orice tip de argumente.
- **GREATEST(expr1 [,expr2]...)** – returneaza cea mai mare valoare din lista de argumente numerice, sau expresia care are prima litera pozitionata ultima in alfabet , pentru date alfanumerice.

SQL> SELECT greatest(12,34,77,89) from dual;

- **LEAST(expr1 [,expr2]...)** – similar cu GREATEST dar returneaza cea mai mica valoare din lista de argumente.
- **DECODE(expr,search1,result1,...default)** - **expr** este comparata cu fiecare valoare **search** si intoarce **rezult** daca **expr** este egala cu valoarea **search,** iar daca nu gaseste nicio egalitate intoarce valoarea **default**.
- Functia DECODE are efectul unui **CASE** sau a unei constructii **IF-THEN-ELSE**.

Functii diverse

- Tipurile de parametri pot fi :
  - **expression** poate fi orice tip de data;
  - **search** este de acelasi tip ca *expression;*
  - **result** este valoarea intoarsa si poate fi orice tip de data;
  - **default** este de acelasi tip ca *result.*

Exemplu:

- In exemplul urmator se calculeaza prima in raport de functia angajatului in companie:

```
SQL>  SELECT nume,functie,salariu,
      decode(functie,'DIRECTOR', salariu*1.35, 'ANALIST',
      salariu*1.25, salariu/5) prima
      FROM angajati WHERE id_dep=20
      ORDER BY  functie;
```

Functii diverse

- **CASE expr WHEN value1/expr1 THEN statements_1;**
  WHEN value2/expr2 THEN statements_2;
  ...
  [ELSE statements_k;]
  END
- CASE functioneaza similar cu functia DECODE. Parametrii sunt:
  - **expr** – este expresia care se va evalua
  - **statements_1** – este valoarea returnata cand **expr = value1/expr1;**
  - **statements_2** – este valoarea returnata cand **expr = value2/expr2;**
  - **statements_k** – este valoarea implicită returnata cand **expr <> value1, value2, ...**

Observatii:

- Expresia expr_i poate fi diferita pentru fiecare ramura WHEN;
- Expresia expr_i poate contine mai multi operatori de comparative;
- Functia CASE poate fi folosita in cererea SELECT sau clauza WHERE si are mai multe tipuri de sintaxa.

Functii diverse

Exemple:

```
SQL>  SELECT nume,data_ang, salariu,
      CASE
      WHEN salariu <= 1000 THEN salariu*0.1
      WHEN salariu > 1000 THEN salariu*0.2
      END prima
      FROM angajati;

SQL>  SELECT id_ang, nume,functie, data_ang, salariu
      FROM angajati
      WHERE functie= (CASE id_ang
                        WHEN 7839 then 'PRESEDINTE'
                        WHEN 7698 then 'DIRECTOR'
                        END);
```

Functii diverse

- **NVL(expr1 ,expr2)** – returneaza **expr2** daca **expr1** este **null**

```
SQL> SELECT nume,data_ang,comision,nvl(comision,0) nvl_com
     FROM angajati WHERE id_dep=30
     ORDER BY nume;
```

- **COALESCE(expr1 ,expr2, ...)** – returneaza prima expresie **not null** din lista de argumente

```
SQL> SELECT nume,data_ang, comision, coalesce(comision,0)
     FROM angajati WHERE id_dep=30
     ORDER BY nume;
```

Functii diverse

- De retinut ca valoarea null trebuie obliglatoriu convertita la zero atunci cand se fac operatii aritmetice, altfel rezultatul va fi null.
- Functiile pot fi imbricate pana la orice nivel, ordinea de executie fiind din interior spre exterior.

USER – returneaza userul curent Oracle

Functii de grup

➢Functii de grup

- Sintaxa pentru functiile de grup este urmatoarea:

SELECT [column,] group_function(column)...

FROM table

[WHERE condition]

[GROUP BY [CUBE] [ROLLUP] group_expression]

[HAVING having_expression]

[ORDER BY column(position)] [ASC,DESC];

Functii de grup

- Parametrii comenzii au urmatoarea semnificatie(cei din paranteze sunt optionali):
- ***group_function*** – specifica numele functiei/functiilor de grup;
  - ***WHERE condition*** – reprezinta o inlantuire de conditii care trebuie sa fie indeplinita in criteriul de selectie a liniilor***;***
  - ***GROUP BY group_expresion*** - specifica coloanele dupa care se face gruparea rezultatelor;
  - ***HAVING***– este folosita pentru a specifica conditiile de grupare a rezultatelor;
  - ***having_expresion*** – reprezinta un nume de coloana sau o expresie care poate folosi functii si coloane;
  - ***ORDER BY***– specifica coloanele(sau ordinea coloanelor specificate prin pozitie) dupa care se face ordonarea rezultatelor;

Functii de grup

- ***CUBE*** – este un operator folosit impreuna cu o funcţie de grup pentru a genera randuri suplimentare in rezultate(produce subtotaluri pentru toate combinatiile posibile ale gruparii specificate in clauza GROUP BY);

- ***ROLLUP*** – este un operator folosit pentru a obtine un set de rezultate care contin subtotaluri, pe langa valorile pentru randurile grupate in mod obisnuit.

Functii de grup

- Aceste functii returneaza rezultate bazate pe grupuri de inregistrari, nu pe o singura inregistrare ca cele prezentate la punctele anterioare.
- Gruparea se face folosind clauza GROUP BY intr-o cerere SELECT si in acest caz toate elementele listei trebuie cuprinse in clauza de grupare.
- Functiile de grup pot fi apelate si in clauza HAVING, dar nu pot fi apelate in clauza WHERE in mod explicit.
- Se poate folosi operatorul DISTINCT pentru a sorta numai elementele distincte din lista, dar se poate folosi si operatorul ALL pentru a considera si inregistrarile duplicate.

Functii de grup

- **AVG([DISTINCT/ALL] expr)** – returneaza valoarea medie a lui **expr**, ignorand valorile nule.

Exemplu: Sa calculam valoarea medie a salariului si comisionului pe toate departamentele:

```
SQL>  SELECT avg(salariu), avg(comision) FROM
      angajati;
```

Exemplu: Daca dorim sa aflam valorile medii pe fiecare departament, trebuie sa folosim obligatoriu clauza GROUP BY:

```
SQL>  SELECT id_dep, avg(salariu), avg(comision)
      FROM angajati
      GROUP BY id_dep;
```

- Prin folosirea operatorilor DISTINCT si ALL rezultatul interogarii poate fi diferit:

```
SQL>  SELECT id_dep, avg(salariu), avg(all salariu),
       avg(distinct salariu)
       FROM angajati GROUP BY id_dep;
```

Functii de grup

- **COUNT(* | [DISTINCT/ALL] expr)** – returneaza numarul de linii intoarse de interogare. Daca se foseste * se numara toate liniile, inclusiv cele care contin valori nule pe anumite coloane, iar daca se foloseste **expr** se numara numai valorile **not null** ale lui **expr**.

Exemplu: Pentru a afla numarul angajatilor care au primit comision pe fiecare departament, folosim urmatoarea cerere:

```
SQL> SELECT id_dep, count(*), count(comision),
        count(all comision), count(distinct comision)
        FROM angajati GROUP BY id_dep;
```

- **MAX([DISTINCT/ALL] expr)** – returneaza valoarea maxima pentru **expr**.

Exemplu: Sa calculam salariul maxim pe fiecare departament:

```
SQL> SELECT a.id_dep, b.den_dep, max(salariu)
        FROM angajati a, departamente b
        WHERE a.id_dep=b.id_dep
        GROUP BY a.id_dep, b.den_dep;
```

Functii de grup

- **MIN([DISTINCT/ALL] expr)** – returneaza valoarea minima pentru **expr.**

Exemplu: Sa calculam venitul minim pe fiecare departament:

```
SQL>  SELECT id_dep,
          min(salariu + nvl(comision,0)) venit_minim
          FROM angajati  GROUP BY id_dep;
```

- Trebuie mentionat ca operatorii DISTINCT si ALL nu au vreun efect pentru functiile MIN si MAX.
- **SUM([DISTINCT/ALL] expr)** – returneaza suma valorilor pentru **expr.**

Exemplu: Suma salariilor si comisioanelor pe fiecare departament:

```
SQL>  SELECT id_dep, sum(salariu),
          sum(distinct salariu), sum(comision)
          FROM angajati GROUP BY id_dep;
```

Functii de grup

- **CUBE**– Genereaza un superset de grupuri prin referiri incrucisate pe coloanele incluse in clauza GROUP BY.

Exemplu: Cererea urmatoare calculeaza salariul total pe toate departamentele, pe fiecare tip de functie, pe fiecare departament si pe fiecare tip de functie din cadrul departamentelor 10 si 20:

```
SQL> SELECT id_dep, functie, SUM(salariu) salariu_total
     FROM angajati
     WHERE id_dep < 30
     GROUP BY CUBE (id_dep, functie);
```

ID_DEP	FUNCTIE	SALARIU_TOTAL
		19625
	ANALIST	6000
	DIRECTOR	5425
	TEHNICIAN	3200
	PRESEDINTE	5000
10		8750
10	DIRECTOR	2450
10	TEHNICIAN	1300
10	PRESEDINTE	5000
20		10875
20	ANALIST	6000
20	DIRECTOR	2975
20	TEHNICIAN	1900

Functii de grup

- **ROLLUP** – Genereaza un superset de grupuri pe coloanele incluse in clauza GROUP BY.

Exemplu: Cererea urmatoare calculeaza salariul total pe fiecare tip de functie din cadrul departamentelor, pe fiecare departament si pe toate departamentele care indeplinesc restrictia de id_dep(10 si 20):

```
SQL> SELECT id_dep, functie, SUM(salariu) salariu_total
     FROM angajati
     WHERE id_dep < 30
     GROUP BY ROLLUP (id_dep, functie);
```

ID_DEP	FUNCTIE	SALARIU_TOTAL
10	DIRECTOR	2450
10	TEHNICIAN	1300
10	PRESEDINTE	5000
10		8750
20	ANALIST	6000
20	DIRECTOR	2975
20	TEHNICIAN	1900
20		10875
		19625

Subcereri SQL

Alexandru Boicea Curs: Baze de date

Subcereri SQL

- In Oracle SQL*Plus subcererile sunt cereri SQL incluse in clauzele SELECT, FROM, WHERE, HAVING sau ORDER BY ale altei cereri numite si cerere principala.
- Rezultatele returnate de o subcerere sunt folosite de o alta subcerere sau de cererea principala in situatii cum ar fi:
  - Crearea de tabele sau view-uri;
  - Inserarea, modificarea si stergerea inregistrarilor din tabele;
  - In furnizarea valorilor pentru conditii puse in comenzile SELECT, UPDATE, DELETE, INSERT si CREATE TABLE.

Subcereri SQL

- Din punct de vedere al rolului pe care il au intr-o comanda SQL si a modului de a face constructia comenzii, subcererile pot fi:

  - **Subcereri ascunse**
  - **Subcereri corelate**
  - **Subcereri pe tabela temporara**
  - **Subcereri pe clauza HAVING**
  - **Subcereri pe clauza SELECT**
  - **Subcereri pe clauza ORDER BY**

Subcereri ascunse

➢ Subcereri ascunse

- Subcererile ascunse pot fi impartite in mai multe categorii, in functie de numarul de coloane sau linii pe care le returneaza:
  - **Subcereri care returneaza o valoare**
  - **Subcereri care returneaza o coloana**
  - **Subcereri care returneaza o linie**
  - **Subcereri care returneaza mai multe linii**

✓ **Subcereri care returneaza o valoare**

- Aceste subcereri se folosesc intr-o alta cerere si au urmatoarea constructie:

```
SELECT column 1, column 2, ...
FROM  table 1
WHERE column =
            ( SELECT column  FROM table 2
              WHERE condition );
```

Subcereri ascunse

- Subcererea se executa prima pentru a verifica conditia din clauza WHERE si poate intoarce o singura valoare dintr-o tabela, prin conditii care depind de datele din tabela folosita in cererea principala. Daca conditia din clauza WHERE a subcererii nu returneaza o singura valoare, atunci se va genera o eroare.

Exemplu:

- Pentru a afla care sunt angajatii care au cel mai mare salariu in firma, putem sa folosim urmatoarea cerere:

```
SQL>  SELECT id_dep, nume, functie, salariu
      FROM angajati
      WHERE salariu=( SELECT max(salariu)
                      FROM angajati );
```

Subcereri care returneaza o coloana

✓ **Subcereri care returneaza o coloana**

- Sunt subcereri care returneaza mai multe valori si folosesc operatorii **IN** si **NOT IN** intr-o constructie ca mai jos:

```
SELECT column 1, column 2, ...
FROM  table 1
WHERE column  IN [NOT IN]
                ( SELECT column
                  FROM table 2
                  WHERE condition );
```

- In cazul in care am folosi operatorul **=** in locul operatorului **IN** s-ar genera o eroare pentru ca subcererea returneaza mai mult de o valoare (ORA-01427). Trebuie specificat ca operatorul **IN** se poate folosi in locul operatorului **=** , dar invers este gresit.

Subcereri care returneaza o coloana

Exemplu:

- Daca vrem sa aflam salariile angajatilor care au functii similare functiilor aferente departamentului 20, folosim urmatoarea cerere :

```
SQL> SELECT nume, functie, salariu
     FROM angajati
     WHERE functie IN
          ( SELECT DISTINCT functie
            FROM angajati
            WHERE id_dep=20);
```

Subcereri care returneaza o linie

✓ Subcereri care returneaza o linie

- Sunt subcereri care returneaza mai multe coloane dar o singura linie si folosesc operatorii **IN** si **NOT IN,** intr-o constructie ca mai jos:

```
SELECT column 1, column 2, ...
FROM  table 1
WHERE  (column 1,column 2...) = [<>][IN] [NOT IN]
                ( SELECT column 1,column 2..
                  FROM table 2
                  WHERE condition );
```

- Numarul, ordinea si tipul coloanelor din clauza WHERE trebuie sa coincida cu cele din subcerere.

Subcereri care returneaza o linie

Exemplu:

- Daca dorim care angajati din departamentul 30 au aceeasi functie ca cea a lui Tache Ionut, folosim urmatoarea cerere:

```
SQL>  SELECT id_dep, nume, functie, data_ang
      FROM angajati
      WHERE (id_dep, functie) =
            (SELECT id_dep, functie FROM angajati
             WHERE nume='TACHE IONUT');
```

- In acest caz subcererea returneaza o singura linie, deoarece Tache are o singura functie si este angajat la un singur departament. In eventualitatea ca pot exista mai multi angajati cu acelasi nume, se va folosi clauza DISTINCT in subcerere(daca au acelasi depart si job).

Subcereri care returneaza mai multe linii

✓ **Subcereri care returneaza mai multe linii**

- Aceste subcereri au o constructie asemanatoare ca la punctul precedent, dar intorc mai multe linii cu mai multe coloane, drept pentru care se mai numesc si subcereri care intorc o tabela.

```
SELECT expr 1,... expr n
FROM  table 1
WHERE  (expr 1,...expr k)  IN [NOT IN]
                ( SELECT expr 1, ...expr k
                  FROM table 2
                  WHERE condition );
```

- Trebuie specificat ca in locul coloanelor se pot folosi si expresii. Numarul de expresii si ordinea lor specificate in clauza WHERE trebuie sa coincida cu numarul de expresii din subcecere si sa fie de acelasi tip.

Subcereri care returneaza mai multe linii

Exemplu:

- Pentru a afla persoanele care au venit maxim pe fiecare departament, folosim urmatoarea cerere:

```
SQL>  SELECT id_dep, nume, functie, salariu+nvl(comision,0) venit
      FROM angajati
      WHERE (id_dep, salariu+nvl(comision,0)) IN
        (SELECT id_dep,max(salariu+nvl(comision,0)) venit_max
         FROM angajati GROUP BY id_dep);
```

- In cazul unei subcereri ascunse, cererea SELECT din subcerere ruleaza prima si se executa o singura data, intorcand valori ce vor fi folosite de cererea principala.

Subcereri corelate

➢ **Subcereri corelate**

- Subcererile corelate se executa o data pentru fiecare linie candidat prelucrata de cererea principala si la executie folosesc cel putin o valoare dintr-o coloana din cererea principala.
- O subcerere corelata se relationeaza cu cererea exterioara prin folosirea unor coloane ale cererii exterioare in clauza predicatului cererii interioare.

Constructia unei subcereri corelate este urmatoarea:

```
SELECT expr 1,... expr n
FROM  table 1
WHERE  (expr 1,...expr k)  IN [NOT IN]
            ( SELECT expr 1, ...expr k
              FROM table 2
              WHERE table 2.expr x = table 1.expr y  [AND,OR] .. );
```

Subcereri corelate

- Pentru o cerere corelata, subcererea se executa de mai multe ori, cate o data pentru fiecare linie prelucrata de cererea principala, deci cererea interioara este condusa de cererea exterioara.
- Pasii de executie ai unei subcereri corelate sunt:
  - se obtine linia candidat prin procesarea cererii exterioare;
  - se executa cererea interioara corelata cu valoarea liniei candidat;
  - se folosesc valorile rezultate din cererea interioara pentru a prelucra linia candidat;
  - se repeta pana nu mai ramane nicio linie candidat.
- Trebuie specificat ca desi subcererea corelata se executa repetat, aceasta nu inseamna ca subcererile corelate sunt mai putin eficiente decat subcererile ascunse.

Subcereri corelate

Exemplu:

- Daca vrem sa aflam persoanele care au salariul peste valoarea medie a salariului pe departamentul din care fac parte, folosim cererea urmatoare:

```
SQL>  SELECT a.id_dep, a.nume, a.functie, a.salariu
        FROM angajati a WHERE a.salariu >
            (SELECT avg(salariu) salariu_mediu FROM angajati b
             WHERE b.id_dep=a.id_dep )
             ORDER By id_dep;
```

- Cererea principala proceseaza fiecare linie din tabela angajati si pastreaza toti angajatii care au salariul peste salariul mediu pe departamentul respectiv returnat de subcerere, care se coreleaza cu cererea principala prin conditia din clauza WHERE. In acest exemplu se foloseste un join pe aceeasi tabela.

Subcereri imbricate

- Subcererile pot fi imbricate. De exemplu, pentru a afla care angajati din firma au salariul mai mare decat salariul maxim din departamentul de proiectare, folosim cererea urmatoare:

```
SQL> SELECT nume, functie, data_ang, salariu
        FROM angajati
        WHERE salariu > ( SELECT max(salariu)
                          FROM angajati
                          WHERE id_dep= (SELECT id_dep
                                         FROM departamente
                                         WHERE
                                           den_dep= 'PROIECTARE'));
```

Subcereri pe tabela temporara

➢ Subcereri pe tabela temporara

- Aceste subcereri se intalnesc in cazul in care se foloseste o subcerere la nivelul clauzei FROM din cererea principala. Constructia este urmatoarea:

```
SELECT  t1.column1,t1.column2,.. t2.expr 1,... t2.expr k
FROM    table t1,
            ( SELECT expr 1, ...expr n
              FROM table
              WHERE conditions ) t2
WHERE conditions
ORDER BY expr;
```

Subcereri pe tabela temporara

Exemplu:

- Pentru a afla salariul maxim pe fiecare departament, se poate face o constructie ca mai jos, unde subcererea intoarce o tabela temporara cu salariul maxim pe fiecare departament:

```
SQL> SELECT b.id_dep, a.den_dep, b.sal_max_dep
     FROM departamente a,  (SELECT  id_dep, max(salariu)
                  sal_max_dep FROM angajati  GROUP BY id_dep) b
     WHERE a.id_dep = b.id_dep
     ORDER BY b.id_dep;
```

Subcereri pe clauza HAVING

➢ **Subcereri pe clauza HAVING**

- Aceste subcereri sunt specificate in clauza HAVING intr-o constructie ca cea de mai jos :

```
SELECT expr 1,... expr n
FROM  table 1
WHERE conditions
HAVING   expr (operator)
                ( SELECT expr 1, ...expr k
                  FROM table 2
                  WHERE conditions )
GROUP BY expresion;
```

- Trebuie retinut ca o clauza HAVING se foloseste intotdeauna impreuna cu clauza GROUP BY si intr-o clauza HAVING se pot folosi functii de grup in mod explicit.

Subcereri pe clauza HAVING

Exemplu:

- Pentru a afla ce departamente au cei mai multi angajati pe aceeasi functie, folosim urmatoarea cerere:

```
SQL> SELECT d.den_dep, a.functie, count(a.id_ang) nr_ang
        FROM  angajati a, departamente d
        WHERE  a.id_dep = d.id_dep
        HAVING count(a.id_ang) = (SELECT max(count(id_ang))
                                  FROM angajati GROUP BY id_dep, functie)
        GROUP BY d.den_dep, a.functie;
```

- Clauza HAVING poate fi folosita si intr-o subcerere pe tabela temporara.

Subcereri pe clauza SELECT

➢ Subcereri pe clauza SELECT

- Se poate construi o subcerere si pe clauza SELECT, avand urmatoarea constructie :

```
SELECT expr 1,... expr n,
       ( SELECT expr  FROM table 2
         WHERE table 2.expr x = table 1.expr y  [AND,OR] .. ) alias
         FROM  table 1
WHERE  conditions
ORDER BY expr 1,...expr k
```

- Aceste subcereri pot fi corelate sau ascunse dar trebuie sa returneze intotdeauna o singura valoare.

Subcereri pe clauza SELECT

Exemplu:

- Daca vrem sa aflam care sunt sefii directi ai angajatilor din departamentul 20, folosim cererea urmatoare:

```
SQL> SELECT nume nume_ang,
                (SELECT nume FROM angajati b
                 WHERE b.id_ang=a.id_sef) nume_sef
     FROM angajati a
     WHERE id_dep=20
     ORDER BY nume ;
```

Subcereri pe clauza ORDER BY

➢ **Subcereri pe clauza ORDER BY**

- O subcerere pe clauza ORDER BY se foloseste numai pentru cereri corelate si trebuie sa returneze intotdeauna o singura valoare, avand urmatoarea constructie :

```
SELECT expr 1,... expr n,
FROM  table 1
WHERE  conditions
ORDER BY
        ( SELECT expr  FROM table 2
         WHERE table 2.expr x = table 1.expr y  [AND,OR] .. )
         ASC/DESC
```

Subcereri pe clauza ORDER BY

Exemplu:

- Daca vrem sa facem o lista cu angajatii din departamentele 10 si 20, ordonati descrescator tinand cont de numarul lor pe fiecare departament, folosim cererea urmatoare:

```
SQL> SELECT id_dep, nume, functie FROM angajati a
        WHERE id_dep in (10,20)
        ORDER BY
              (SELECT count(*)
               FROM angajati b
               WHERE  a.id_dep=b.id_dep) DESC;
```

Operatori in subcereri

➢ Operatori in subcereri

- Operatorii prezentati pentru cereri sunt valabili si pentru subcereri, dar mai sunt si alti operatori specifici.

✓ **Operatorii SOME/ANY si ALL**

- Acesti operatori sunt folositi in subcereri care intorc mai multe linii si sunt folositi impreuna cu operatorii logici in clauzele WHERE si HAVING.
- Operatorul SOME (sau sinonimul lui ANY) compara o expresie cu fiecare valoare returnata de o subcerere si pastreaza liniile unde expresia indeplineste conditia impusa de operatorul logic.
- Daca este folosit impreuna cu operatorul logic > , atunci are semnificatia de *mai mare decat minim.*

Operatori in subcereri

Exemplu:

- Pentru a afla care sunt angajatii care au salariul mai mare decat cel mai mic salariu pentru functia de programator, folosim cererea urmatoare:

```
SQL> SELECT id_dep, nume, functie, salariu
        FROM angajati
        WHERE salariu > SOME (SELECT DISTINCT salariu
                              FROM angajati
                              WHERE functie='PROGRAMATOR')
        ORDER BY id_dep, nume;
```

Operatori in subcereri

- Operatorul ALL lucreaza similar cu operarorii SOME/ANY, iar daca este folosit impreuna cu operatorul logic **>** , atunci are semnificatia de *mai mare decat maxim.*

```
SQL> SELECT id_dep, nume, functie, salariu
        FROM angajati
        WHERE salariu > ALL ( SELECT DISTINCT salariu
                              FROM angajati
                              WHERE functie='PROGRAMATOR')
        ORDER BY id_dep, nume;
```

Operatori in subcereri

✓ **Operatorii EXISTS si NOT EXISTS**

- Acesti operatori sunt folositi adesea in subcereri corelate si testeaza daca subcererea returneaza cel putin o valoare pentru EXISTS, sau niciuna in cazul lui NOT EXISTS, returnad TRUE sau FALSE.

Exemplu:

- Pentru a afla care departamente au cel putin un angajat, folosim cererea urmatoare:

```
SQL> SELECT d.id_dep, d.den_dep
        FROM  departamente d
        WHERE  EXISTS  ( SELECT nume
                              FROM angajati
                              WHERE id_dep=d.id_dep)
        ORDER BY id_dep;
```

Operatori in subcereri

- Trebuie specificat ca o constructie cu EXISTS este mult mai performanta decat o constructie cu IN, SOME/ANY sau ALL, deoarece in cazul in care folosim tabele temporare acestea nu sunt indexate, ducand la scaderea considerabila a performantelor.
- Performanta depinde de folosirea indecsilor, de dimensiunea tabelelor din baza de date, de numarul de linii returnate de subcerere si daca sunt necesare tabele temporare pentru a evalua rezultatele returnate.
- Desi o subcerere cu o constructie pe operatorul NOT IN poate fi la fel de eficienta ca si in cazul unei constructii pe NOT EXISTS, cea din urma este totusi mult mai sigura, daca subcererea intoarce si valori NULL.

Operatori in subcereri

- In cazul operatorului NOT IN, conditia se evalueaza la FALSE cand in lista de comparatii sunt incluse valori NULL.

Exemplu:

- Daca vrem sa facem o lista cu angajatii care nu sunt sefi si folosim o constructie cu NOT IN, cererea nu va intoarce nicio inregistrare, ceea ce este fals:

```
SQL> SELECT id_dep, id_ang, nume, functie, id_sef
     FROM angajati a
     WHERE id_ang NOT IN ( SELECT DISTINCT id_sef
                           FROM angajati)
     ORDER BY id_dep;
```

- **no rows selected**

Actualizarea datelor prin subcereri

Exemple:

- Urmatoarea comanda actualizeaza comisionul angajatilor la 10% din salariul minim din departamentul din care fac parte, dar numai pentru angajatii care nu au primit comision:

```
SQL>UPDATE angajati a
        SET a.comision=(SELECT min(salariu)*0.1  FROM angajati b
                WHERE a.id_dep=b.id_dep)
        WHERE nvl(comision,0)=0 ;
```

- Pentru a crea o tabela cu angajatii care au ecusonul mai mic decat cel mai mic ecuson de sef pe fiecare departament folosim comanda:

```
SQL> CREATE TABLE ecusoane_old (id_dep,id_sub,nume_sub, id_sef) as
     SELECT id_dep, id_ang, nume,id_sef  FROM  angajati a
     WHERE a.id_ang <(SELECT min(b.id_sef)
                FROM angajati b WHERE b.id_dep = a.id_dep);
```

- Pentru a face o copie a tabelei angajati executam comanda:

```
SQL> CREATE TABLE angajati_copy AS SELECT * FROM angajati;
```

Actualizarea datelor prin subcereri

- Daca vrem sa stergem angajatii care nu sunt sefi din tabela copie, executam urmatoarea comanda:

```
SQL> DELETE FROM angajati_copy
        WHERE id_ang NOT IN (SELECT DISTINCT id_sef FROM angajati
                                WHERE   id_sef  IS not null);
```

8 rows deleted.

Observatie:

- Daca nu se pune conditia in subcerere ca ecusonul sefului sa nu fie null, rezultatul va fi eronat:

```
SQL> DELETE FROM angajati_copy
        WHERE id_ang NOT IN (SELECT DISTINCT id_sef FROM angajati);
```

0 rows deleted.

- Rezultatul eronat este din cauza ca exista un angajat care nu are sef, adica are valoarea NULL pentru ID_SEF (presedintele companiei).

Reguli in subcereri

- Cand folosim subcereri, trebuie sa respectam cateva reguli:
  - Cererea interioara trebuie sa fie inclusa intre paranteze si trebuie sa fie in partea dreapta a conditiei;
  - Expresiile din lista de expresii a subcererii trebuie sa fie in aceeasi ordine ca cele din lista din clauza WHERE a cererii principale (avand acelasi tip si numar de expresii);
  - Subcererile nu pot fi ordonate, deci nu contin clauza ORDER BY;
  - Clauza ORDER BY poate sa fie pusa la sfarsitul cererii principale;
  - Subcererile sunt executate de la cea mai adanca imbricare pana la nivelul principal de imbricare(cu exceptia cererilor corelate);
  - Subcererile pot folosi functii de grup si clauza GROUP BY ;
  - Subcererile pot fi inlantuite cu predicate multiple AND sau OR in aceeasi cerere externa;
  - In subcereri se pot folosi operatori de multimi;
  - Subcererile pot fi imbricate pana la nivelul 255.

Structuri de stocare a datelor

Alexandru Boicea Curs: Baze de date

Structuri de stocare a datelor

- Comenzile de definire a datelor sunt oferite de catre limbajul **DDL** (*Data Definition Language*).

  Aceste comenzi acorda facilitati pentru:

  - Crearea, modificarea si stergerea tabelelor in baza de date;
  - Crearea constrangerilor de integritate si a relatiilor dintre obiectele bazei de date;
  - Definirea vederilor pe tabelele bazei de date;
  - Crearea si administrarea indecsilor;

Crearea si definirea stucturilor tabelare

➢ Crearea si definirea structurilor tabelare

- Comenzile pentru crearea si definirea de structuri tabelare sunt comenzi de tip DDL si permit crearea, definirea de constrangeri si relationarea lor intr-o baza de date.
- Structura unei tabele este data de urmatoarele specificatii de definire:
  - definirea coloanelor;
  - definirea constrangerilor de integritate;
  - definirea tablespace-lui unde se creeaza;
  - definirea parametrilor de stocare;

Crearea si definirea stucturilor tabelare

- Sintaxa comenzii de creare a unei tabele este urmatoarea(parametrii dintre paranteze drepte sunt optionali):

CREATE TABLE [schema.] table_name

column _name

column datatype [DEFAULT expr]

[column_constraints]

[table_constraints][TABLESPACE tablespace]

[storage parameters]

[ENABLE/DISABLE clause] [AS subquery]

Crearea si definirea stucturilor tabelare

unde :

- **schema** – este schema unde se creeaza tabela (specifica userul si baza de date);
- **table_name** – este numele tabelei;
- **column** – este numele coloanei;
- **datatype** – reprezinta tipul coloanei;
- **DEFAULT expr** – specifica valoarea implicita a coloanei;
- **column_constraints** – defineste constrangerile de integritate pentru coloana;
- **table_constraints** - defineste constrangerile de integritate la nivel de tabela;
- **tablespace** – specifica in ce tablespace al bazei de date se creeaza tabela.

Crearea si definirea stucturilor tabelare

- **storage parameters** – defineste parametrii de creare si pot fi:
  - PCTFREE –procentaj de spatiu rezervat pentru update;
  - PCTUSED – procent minim folosit pentru un bloc de date;
  - INITRANS – numarul initial de tranzactii pentru fiecare bloc(1-255);
  - MAXTRANS- numarul maxim de tranzactii concurente (1-255);
  - CLUSTER – specifica daca tabela face parte dintr-un cluster.
- **ENABLE/DISABLE clause** – activare/dezactivare de constrangeri;
- **AS subquery** – inserare de date dintr-o alta tabela obtinute printr-o interogare.

Crearea si definirea stucturilor tabelare

- Tipurile de date care pot fi asociate coloanelor unei tabele pot fi :
  - numerice;
  - alfanumerice;
  - data calendaristica si timp;
  - compuse (matrice sau tabela) .
- Cateva dintre cele mai uzuale tipuri sunt:
- **NUMBER** – numar real de dimensiune variabila (maxim 38 cifre);
- **NUMBER(n)** – numar intreg de **n** cifre;
- **NUMBER(n,m)** – numar real de **n** cifre din care **m** zecimale;
- **CHAR(n)** – sir de caractere de lungime fixa **n** (1- 2000);

Crearea si definirea stucturilor tabelare

- **NCHAR(n)** – analog cu CHAR dar poate stoca siruri de caractere nationale;
- **VARCHAR2(n)** – sir de caractere de lungime variabila **n** (1-4000);
- **NVARCHAR2(n**) – analog cu VARCHAR dar poate stoca siruri de caractere;
- **LONG** – sir de caractere de maxim 2 la puterea 31 octeti;
- **LONG RAW** – similar cu LONG dar contine date binare ;
- **ROWID** – poate stoca identificatorul unei linii din tabela;
- **DATE** – data calendaristica;
- **TIMESTAMP(n)** – extensie pentru tipul DATE si contine si fractiuni de secunda pe **n** zecimale .

Crearea si definirea stucturilor tabelare

- Cand se creeaza o tabela trebuie respectate anumite regului cum ar fi :
  - Userul trebuie sa aiba drept de creare de tabele (privilegiul CREATE TABLE);
  - Numele tabelei trebuie sa fie unic in contul in care se creeaza si nu este case sensitive;
  - Numele trebuie sa aiba maxim 30 caractere continue, sa inceapa cu o litera si sa nu fie cuvant rezervat Oracle;
  - O tabela poate fi creata oricand si nu poate fi alterata cand este accesata de un alt user;
  - La creare nu trebuie specificata dimensiunea tabelei dar trebuie estimat ce spatiu va ocupa in tablespace;
  - Structurile tabelare pot fi modificate si ulterior(adaugare sau stergere de coloane, constrangeri, indecsi, etc).

Crearea si definirea stucturilor tabelare

- Daca dimensiunea initiala este insuficienta i se aloca automat mai mult spatiu in limita tablespace-lui (care la randul lui poate fi extins cu un nou data_file);
- Clauza DEFAULT poate contine constante numerice, sir de caractere , functii (inclusiv *sysdate* si *user*) dar nu poate contine numele unei alte coloane sau pseudocoloane);
- Valoarea DEFAULT este luata in considerare in cazul inserarii unei linii in care nu se specifica nimic in legatura cu coloana respectiva.

Crearea si definirea stucturilor tabelare

Exemplu:

```
SQL>  CREATE TABLE studenti
      (facultate char(30)  DEFAULT  'Automatica si Calculatoare',
      catedra char(20),
      CNP number(13),
      nume    varchar2(30),
      data_nastere date,
      an_univ number(4)  DEFAULT  2014,
      media_admitere number(5,2) ,
      discip_oblig  varchar2(20)  DEFAULT 'Matematica',
      discip_opt    varchar2(20)  DEFAULT 'Fizica' ,
      operator  varchar2(20)  DEFAULT  user,
      data_op   date DEFAULT sysdate );
```

- In cazul inserarii unei linii, valorile campurilor operator si data operare nu trebuie specificate deoarece sunt preluate prin functiile sistem *user* si *sysdate*.

Crearea si definirea stucturilor tabelare

✓ **Crearea unei tabele printr-o cerere SELECT**

- In acest caz tabela va fi creata in urma unei cereri SELECT si va contine si liniile returnate de cerere. Sunt doua posibilitati de a crea o tabela folosind o astfel de constructie :
- Comanda CREATE TABLE nu contine descrierea structurii tabelei si in acest caz :
  - Se creeaza o tabela cu o structura identica cu campurile specificate in cererea SELECT;
  - Tipurile coloanelor se pastreaza;
  - Daca cererea contine o expresie sau functie trebuie sa i atribuie un alias valid;
  - Noua tabela nu mosteneste nicio constrangere de integritate de la vechile tabele(cu exceptia celei not null).

Crearea si definirea stucturilor tabelare

Exemplu:

- Sa cream o noua tabela care va contine veniturile angajatilor din departamentul 20, folosind tabela angajati:

```
SQL> CREATE TABLE depart_20  AS
        SELECT id_ang, nume, data_ang,
                    salariu+nvl(comision,0) venit
         FROM angajati  WHERE id_dep=20;
```

- Noua tabela DEPART_20 va avea urmatoarea structura :

Name	Null?	Type
ID_ANG	NOT NULL	NUMBER(4)
NUME		VARCHAR2(17)
DATA_ANG		DATE
VENIT		NUMBER

- Cu toate ca **ID_ANG** este creata cu optiunea **NOT NULL** coloana nu este definita ca o cheie primara sau unica.

Crearea si definirea stucturilor tabelare

Exemplu:

- Sa cream o tabela cu primele angajatilor din departamentul 30 daca prima reprezinta 15% din venitul lunar:

```
SQL> CREATE TABLE depart_30
       (depart    DEFAULT 30,
        nume      NOT NULL,
       data_ang,
       prima   )
        AS
            SELECT  id_dep, nume, data_ang,
                        round(0.15*(salariu + nvl(comision,0)),2)
            FROM angajati
            WHERE id_dep=30;
```

Constrangeri de integritate

➢ Constrangeri de integritate

- Constrangerile de integritate sunt restrictii care trebuie respectate la nivel de tabela sau in relatiile cu alte tabele ;
- Aceste reguli sunt verificate automat in cazul operatiilor de inserare, stergere, modificare si, in cazul in care nu se valideaza, se genereaza o eroare si tranzactia este anulata.
- Constrangerile de integritate pot fi:
  - **NOT NULL** - inregistrarile nu pot contine valori nule;
  - **UNIQUE** - defineste o valoare unica pe una sau mai multe coloane (nu pot fi mai multe inregistrari cu aceleasi valori pe coloanele respective, dar accepta valori nule);
  - **PRIMARY KEY** - defineste o cheie primara pe o coloana sau coloane multiple (nu pot fi mai multe inregistrari cu aceeasi cheie primara);
  - **FOREIGN KEY** - defineste o cheie externa (tabela se relationeaza cu alta tabela pe o cheie primara, cheie unica sau coloane cu unicitate);
  - **CHECK** - forteaza o conditie pe coloana, conditia putand sa contina si functii (cu unele exceptii, de exemplu sysdate si user).

Constrangeri de integritate

- Caracteristici ale constrangerilor de integritate:
  - Fiecare constrangere va avea un nume dat de user sau generat automat de catre sistem ;
  - Constrangerile pot fi activate sau dezactivate cu comanda ALTER TABLE ;
  - Constrangerile pot fi adaugate sau sterse si dupa ce o tabela a fost creata ;
  - Informatiile legate de constrangeri se pastreaza in dictionarul de date.

✓ Constrangerea **NOT NULL**:

- Se aplica la nivel de coloane si verifica daca inregistrarile au valorea nula pe coloanele respective, fortand un cod de eroare care anuleaza tranzactia ;
- Cand se creeaza constrangeri pe o cheie primara se creeaza automat si o constrangere NOT NULL pe coloanele respective (o cheie primara nu trebuie sa contina valori nule pe coloanele care o definesc).

Constrangeri de integritate

✓ Constrangerea **UNIQUE**:

- Verifica ca o coloana, sau perechi de coloane, sa nu contina valori duplicate ;
- Verificarea se face numai pentru inregistrarile cu valori nenule deoarece constrangerea permite inserarea de valori nule in coloanele respective. Daca se asociaza cu constrangerea NOT NULL se creeaza cheie unica;
- In mod automat se creeaza si un index pe coloanele definite chei unice, ceea ce duce la marirea vitezei de interogare pe tabela ;
- Daca constrangerea se creeaza pe mai multe coloane atunci setul de date trebuie sa fie unic.
- Sintaxa pentru definirea la nivel de coloana este:

column_name datatype [CONSTRAINT constraint_name] UNIQUE

- Sintaxa pentru definirea la nivel de tabela(ultima linie) este:

[, CONSTRAINT constraint_name] UNIQUE (col1[, col2, [...]])

Constrangeri de integritate

✓ Constrangerea **PRIMARY KEY**:

- Se foloseste pentru definirea cheii primare pe una sau mai multe coloane;
- O tabela poate avea o singura constrangere de tip PRIMARY KEY si nu accepta valori nule pe coloana sau coloanele care o definesc ;
- Cand se creeaza o cheie primara se creeaza in mod automat si o constangere de timp NOT NULL si UNIQUE ;
- Cand se creeaza o cheie primara se creeaza in mod automat si un index pe coloanele care o definesc(care imbunatateste timpul de interogare).
- Sintaxa pentru definirea la nivel de coloana este:

column_name datatype [CONSTRAINT constraint_name] PRIMARY KEY

- Sintaxa pentru definirea la nivel de tabela(ultima linie) este:

[, CONSTRAINT constraint_name] PRIMARY KEY(col1[, col2, [...]])

Constrangeri de integritate

✓ Constrangerea **FOREIGN KEY**:

- Se foloseste pentru a relationa o talela cu una sau mai multe tabele, verificand daca valorile continute in coloanele definite de FOREIGN KEY (numita cheie straina sau cheie externa) sunt cuprinse in valorile coloanelor altei tabele care trebuie sa fie definite ca UNIQUE sau PRIMARY KEY.
- Sintaxa pentru definirea la nivel de coloana este:

column_name datatype [CONSTRAINT constraint_name]

REFERENCES table(column)

[on DELETE CASCADE | DELETE SET NULL]

- Sintaxa pentru definirea la nivel de tabela(ultima linie) este:

[, CONSTRAINT constraint_name] FOREIGN KEY(col1[, col2, [...]])

REFERENCES table(column)

[on DELETE CASCADE | DELETE SET NULL]

Constrangeri de integritate

- Reguli pentru constrangerea FOREIGN KEY:
  - Inserarea unei linii intr-o tabela relationata (pe care am definit FOREIGN KEY) se poate face numai daca exista decat o singura linie in tabela de referinta (in care am definit PRIMARY KEY sau UNIQUE) corespunzator coloanelor de relationare ;
  - Stergerea unei linii din tabela de referinta nu se poate face atata timp cat exista linii relationate pe linia respectiva in tabela relationata ;
  - Regulile de mai sunt valabile si in cazul relationarii pe coloane;
  - Pentru a putea sterge in tabela de referinta linii referite in alte tabele se foloseste optiunea ON DELETE CASCADE. In acest caz, cand se sterge o linie din tabela de referinta se vor sterge toate liniile din tabelele relationate care sunt in relatie cu linia respectiva ;
  - In cazul unei tabele relationata cu ea insasi, stergerea unei linii care este referita duce la aparitia de valori nule pe coloanele din liniile relationate(de exemplu, in tabela angajati, cand se sterge linia aferenta unui sef, toti angajatii care au seful respectiv vor avea valoare null pe coloana id_sef) ;
  - Folosind optiunea ON DELETE SET NULL, coloanele de relatie din tabela relationata devin nule si nu sunt sterse liniile relationate atunci cand se sterge o linie din tabela de referinta.

Trunchierea datelor dintr-o tabela

➢ **Comanda TRUNCATE**

Este folosita pentru trunchierea (golirea de date) a unei tabele si are sintaxa:

TRUNCATE TABLE [schema.] table_name

[DROP STORAGE] [REUSE STORAGE]

unde :

- **schema** – este schema unde este creata tabela (specifica userul si baza de date);
- **table_name** – este numele tabelei;
- **DROP STORAGE** – elibereaza spatiul rezultat din stergerea liniilor;
- **REUSE STORAGE** – pastreaza spatiul rezultat din stergerea liniilor alocat tabelei.

Trunchierea datelor dintr-o tabela

Exemplu:

SQL> TRUNCATE TABLE angajati_copy;

Observatii:

- Trunchierea unei tabele cu optiunea REUSE STORAGE este utila atunci cand se folosesc in mod uzual tabele temporare in diferite prelucrari si exista riscul ca spatiul pe disc sa fie prea mic la o noua rulare. Acest risc apare si atunci cand se sterge o tabela temporara cu comanda DROP si se recreeaza, daca intre timp spatiul pe disc nu mai are dimensiunea necesara numarului de linii rezultate din prelucrari.
- Cand se foloseste comanda TRUNCATE datele nu mai pot fi recuperate cu comanda ROLLBACK, aceasta facilitate este oferita numai in cazul in care se foloseste comanda DELETE pentru stergerea datelor.
- Daca dupa comanda DELETE se da comanda COMMIT datele nu mai pot fi refacute cu comanda ROLLBACK, deoarece tranzactiile au fost executate pe baza de date.

Crearea si utilizarea vederilor

➢ Crearea si utilizarea vederilor

- Vederea (*view*) este un tip de obiect care se poate crea prelucrand date stocate in tabelele bazei de date.
- O vedere se creeaza prin asocierea unei cereri SELECT cu un nume de obiect stocat in baza de date.
- O vedere se comporta ca o tabela in cazul executiei de cereri SELECT si poate fi folosita uneori pentru efectuarea de actualizari ale bazei de date.
- Exista insa si diferente intre o vedere si o tabela, mai ales din punct de vedere al implementarii:

Crearea si utilizarea vederilor

- In cazul unei vederi, baza de date stocheaza definitia acesteia (cererea SQL ca sir de caractere) si nu datele regasite prin aceasta(este o tabela logica, nu fizica, cu exceptia unui view materializat).
- Ori de cate ori se executa o cerere avand la baza o vedere, ea este recalculata. Astfel, orice modificare efectuata in tabelele pe baza carora e definita vederea se reflecta automat in aceasta.
- Structura unei vederi nu se poate modifica prin cereri de tip ALTER ci prin recrearea vederii cu specificarea unei alte cereri SQL.
- Constrangerile de integritate nu se pot defini la nivel de vedere.
- O vedere mosteneste implicit toate constrangerile definite pe coloanele tabelelor pe baza carora este definita prin cererea asociata ei.

Crearea si utilizarea vederilor

- Vederile permit implementarea de scheme externe prin care diverse categorii de utilizatori acceseaza doar acele portiuni din baza de date pe care le folosesc in mod obisnuit, asigurandu-se astfel:
  - Confidentialitatea datelor - utilizatorii care acceseaza baza de date doar prin intermediul unor vederi pot fi restrictionati in ceea ce priveste datele pe care le pot utiliza prin insasi definitia vederilor respective (ele nu contin coloanele/liniile/datele la care nu se doreste accesul acelei categorii de utilizatori).
  - Corectitudinea datelor - prin faptul ca un utilizator nu are acces la date pe care uzual nu le foloseste, impiedicand coruperea accidentala a celorlalte date din cauza necunoasterii semnificatiei lor.

Crearea vederilor

- Sintaxa comenzii de creare a unei vederi este urmatoarea(parametrii dintre parantezele drepte sunt optionali):

  CREATE [OR REPLACE]

  VIEW [schema.] view_name [(column_list)]

  AS query_statement;

unde:

- **view_name** - specifica numele asociat vederii respective. Respecta aceleasi conditii ca in cazul numelor de tabele.
- **query_statement** - cererea SQL de tip SELECT asociata vederii.

Crearea vederilor

- **schema** – este numele schemei(userului) asociata vederii.
- **column_list** - specifica numele coloanelor din vedere. Acestea trebuie sa respecte restrictiile uzuale pentru nume de coloane descrise in capitolele precedente (30 de caractere, etc.). In lipsa acestei optiuni coloanele vederii au aceleasi nume cu ale rezultatului cererii SQL asociate. In cazul in care rezultatul cererii asociate vederii are nume de coloane care nu respecta restrictiile privind astfel de nume, este obligatorie, fie folosirea listei_de_coloane, fie modificarea cererii prin adaugarea unor aliasuri de coloana corespunzatoare.

Crearea vederilor

- **OR REPLACE** - in cazul in care exista deja o vedere cu acel nume, se face suprascriere. Este modul uzual prin care se modifica structura si continutul unei vederi. In lipsa acestei optiuni sistemul semnaleaza in astfel de situatii o eroare.
- **READ ONLY** – specifica ca datele pot fi citite dar nu se permite adaugarea sau modificarea de date.

Crearea vederilor

Exemple:

- Sa cream un view care contine date despre angajatii din departamentul 10:

SQL>CREATE OR REPLACE VIEW angajati_10 AS

SELECT * FROM angajati WHERE id_dep=10;

View created.

- Urmatoarea comanda creeaza un view care contine numai anumite date despre angajati:

SQL>CREATE OR REPLACE VIEW date_angajare AS

SELECT id_ang, nume, data_ang, functie FROM angajati;

View created.

Crearea vederilor

- Un view poate fi creat si prelucrand date din alt view:

```
SQL> CREATE OR REPLACE VIEW joburi AS
        SELECT id_ang ecuson, nume nume_ang, functie job
        FROM date_angajare;
```

View created.

Observatii:

- O vedere se poate folosi pentru definirea altor vederi (JOBURI eate definita pe baza DATE_ANGAJARE).
- Numele coloanelor din vederea JOBURI sunt altele decat cele din tabela ANGAJATI si vederea DATE_ANGAJARE, ca urmare a folosirii unor aliasuri in cererea de creare.

Crearea vederilor

- Se pot defini vederi avand asociate cereri SELECT pe mai multe tabele:

```
SQL>  CREATE OR REPLACE VIEW angajati_dep AS
      SELECT a.den_dep, b.nume, b.data_ang, b.functie job
      FROM departamente a, angajati b
      WHERE a.id_dep=b.id_dep;
```

Observatii:

- In acest trebuie folosite metode de JOIN sau subcereri;
- Astfel de vederi sunt folosite pentru a regasi date si nu pentru modificarea acestora. Ele pot usura dezvoltarea de aplicatii prin utilizarea cererilor care folosesc vederi.

Modificarea datelor in vederi

✓ **Modificarea datelor prin intermediul vederilor**

- Deoarece o vedere nu are date proprii si se calculeaza de fiecare data pe baza cererii SQL asociata, orice modificare este efectuata direct in tabelele bazei de date care sunt specificate in definitia sa. Din aceasta cauza nu orice vedere poate fi folosita pentru modificarea datelor, ci doar cele pentru care sistemul poate detecta exact care linii ale acestora sunt afectate de operatia respectiva.
- In cazul modificarii datelor prin vederi, mecanismele de tranzactie descrise anterior se comporta exact ca in cazul modificarii tabelelor.

Modificarea datelor in vederi

- Comenzile DML trebuie sa respecte un set de restrictii pe care o vedere trebuie sa le indeplineasca pentru a putea fi folosita in operatiile de inserare, modificare si stergere. Acestea sunt diferite dupa tipul operatiei:

✓ **Stergere si actualizare**

- Cererile DELETE si UPDATE se pot aplica pe o vedere daca aceasta nu contine:

  - Functii de grup (SUM, MIN, MAX, COUNT, etc.);
  - DISTINCT;
  - GROUP BY;
  - HAVING;
  - UNION or UNION ALL;

Modificarea datelor in vederi

Exemple:

- Urmatoarele vederi vor da erori la creare si nu pot fi folosite pentru stergerea/actualizarea datelor deoarece incalca una sau mai multe dintre restrictiile de mai sus:

a) Vedere care contine DISTINCT, grupare si functii de grup:

```
SQL>  CREATE OR REPLACE VIEW salarii_max AS
          SELECT DISTINCT id_dep, MAX(salariu)
          FROM angajati
          GROUP BY id_dep;
```

ERROR at line 2:

ORA-00998: must name this expression with a column alias

- Pentru salariul maxim este obligatoriu un alias de coloana deoarece MAX(salariu) nu respecta conventia pentru nume de coloane (contine paranteze).

Modificarea datelor in vederi

b) Vedere care contine literali si join:

```
SQL>  CREATE OR REPLACE VIEW functii_ang AS
        SELECT a.id_dep depart, a.den_dep depart,
                  b.nume nume_ang, b.functie job
        FROM departamente a, angajati b
        WHERE a.id_dep=b.id_dep;
ERROR at line 2:
ORA-00957: duplicate column name
```

- Eroarea apare deoarece coloana DEPART ar duce la crearea unei vederi avand doua coloane cu aceeasi denumire.

Modificarea datelor in vederi

- Prin intermediul vederii SALARII_INDEXATE definita in continuare se pot actualiza toate coloanele vederii care provin din coloane ale tabelei ANGAJATI, dar nu se poate specifica modificarea coloanei suplimentare SALARIU_NOU definita de expresia aritmetica SALARIU*1.1:

```
SQL> CREATE OR REPLACE VIEW salarii_indexate AS
     SELECT id_dep, nume, data_ang, functie,
                 salariu*1.1 salariu_nou
     FROM angajati;
```

1 row updated.

Modificarea datelor in vederi

- Urmatoarea comanda va modifica functia atat in view-ul SALARII_INDEXATE cat si in tabela ANGAJATI:

```
SQL>  UPDATE salarii_indexate
          SET functie='INGINER'
          WHERE nume='TACHE IONUT';
```

- Daca se face insa o modificare de salariu se va genera o eroare deoarece se incearca actualizarea unei coloane din view care a fost definita printr-o expresie:

```
SQL>  UPDATE salarii_indexate
          SET salariu_nou=2500
          WHERE nume='TACHE IONUT'
ERROR at line 2:
ORA-01733: virtual column not allowed here
```

Inserarea datelor in vederi

- **Inserare datelor in vederi**

  Pentru a se putea insera noi linii prin intermediul unei vederi, aceasta trebuie sa satisfaca urmatoarele restrictii:

- Nu exista coloane cu acelasi nume in vedere;
- Vederea nu contine coloane care provin din expresii sau literali (fiecare coloana provine dintr-o coloana a tabelei de baza).
- Inserarea datelor intr-o tabela prin intermediul unei vederi se poate face numai prin vederile create pe o singura tabela, respectand toate constrangerile de integritate ale acesteia.

Inserarea datelor in vederi

Exemple:

- Daca se insereaza o linie noua in view-ul ANGAJATI_10 se va insera automat linia si in tabela ANGAJATI:

```
SQL>  INSERT INTO angajati_10(id_ang, nume, data_ang, functie)
        VALUES (333,'MITRACHE SILVIU', '15-JAN-1986', 'ECONOMIST');
```

- Daca se face o inserare in view-ul SALARII_INDEXATE se va genera o eroare deoarece coloana SALARIU_NOU a fost definite printr-o expresie:

```
SQL> INSERT INTO salarii_indexate(id_dep, nume, data_ang,
                                  functie, salariu_nou)
     VALUES (10,'STROE VICTOR', '1-JUL-1985', 'INGINER', 2300);
ERROR at line 2:
ORA-01733: virtual column not allowed here
```

Stergerea vederilor din dictionar

- Stergerea unei vederi din dictionarul bazei de date se face cu comanda DROP VIEW:

SQL> DROP VIEW angajati_dep;

- Daca se sterge un view din care s-au create alte view-uri, datele se pierd in cascada:

SQL> DROP VIEW date_angajare;

View dropped.

SQL> SELECT * FROM joburi;

ERROR at line 1:

ORA-04063: view "SCOTT.JOBURI" has errors;

- Fiind o comanda DDL, stergerea unei vederi din dictionar nu poate fi anulata cu comanda ROLLBACK.

Crearea si administrarea indecsilor

➢ Crearea si administrarea indecsilor

- Un index este un obiect creat pe baza de date pentru cresterea performantelor de regasire a datelor. Sintaxa comenzii de creare a unui index este urmatoarea:

CREATE

[ONLINE|OFFLINE]

[UNIQUE|FULLTEXT|SPATIAL|BITMAP] INDEX index_name

ON table_name(column_name, ..)

index_options

unde:

- **UNIQUE|FULLTEXT|SPATIAL|BITMAP** – specifica tipul indexului;
- **table_name** – reprezinta numele tabelei pe care se creeaza indexul;
- **column_name** – reprezinta numele coloanelor care compun indexul;
- **index_options** – optiuni care se pot specifica la crearea indexului.

Crearea si administrarea indecsilor

Exemple:

- Crearea unui index de tip arbore(B-Tree) cu unicitate pe id_ang:

SQL> CREATE INDEX scott.angajati_idx ON scott.angajati(id_ang) PCTFREE 30 STORAGE(INITIAL 200K NEXT 200K PCTINCREASE 0 MAXEXTENTS 50) TABLESPACE bd_data;

- Crearea unui index de tip BITMAP pe jobul angajatilor :

SQL> CREATE BITMAP INDEX scott.dep_idx ON scott.angajati(functie) PCTFREE 30 STORAGE(INITIAL 200K NEXT 200K PCTINCREASE 0 MAXEXTENTS 50) TABLESPACE bd_data;

- Tipul indexului este implicit B-Tree si pentru a modifica parametrii unui index se foloseste comanda ALTER INDEX.
- Stergerea unui index din dictionar se face cu comanda DROP INDEX.
- Indecsii se creeaza automat in momentul in care se creeaza o cheie primara sau unicitate pe coloane.
- Unui index i se asociaza automat o tabela index.

Dictionarul bazei de date

Dictionarul bazei de date

➢ **Dictionarul bazei de date**

- Dictionarul bazei de date este o colectie de tabele si view-uri in care sistemul de gestiune stocheaza informatii legate de starea bazei de date.Aceste obiecte sunt create in schema SYS si sunt administrate in exclusivitate de catre sistemul de gestiune(au proprietatea *read only*).
- Datele din dictionar nu pot fi alterate prin comenzi DML.
- Dictionarul stocheaza informatii despre structura logica si fizica a bazei de date, cum ar fi:

  - Obiectele create in baza de date;
  - Constrangerile de integritate create pe obiecte;
  - Userii creati pe baza de date si drepturile de acces acordate;
  - Fisierele de date, de control si de log ale bazei de date;
  - Spatiul fizic si spatiul logic alocate bazei de date;
  - Spatiul alocat si spatiul utilizat de catre useri si obiecte;
  - Statistici create pe baza de date.

Dictionarul bazei de date

➢ **View-ul DICTIONARY**

- View-ul DICTIONARY contine informatii despre toate obiectele bazei de date si o scurta descriere a obiectului.
- Urmatoarea cerere afiseaza continutul tabelei DICTIONARY:

SQL> DESCRIBE dictionary

- Toate tabelele si view-urile dictionarului pot fi vizualizate cu comanda:

SQL> SELECT table_name FROM dictionary;

- Multe dintre vederile dictionarului de date au nume lung. Sinonimele publice au fost furnizate, ca abrevieri, pentru accesul la unele din cele mai comune vederi ale dictionarului. Sursele acestor sinonime se pot vedea cu cererea:

```
SQL> SELECT table_name, comments  FROM  dictionary
     WHERE table_name like 'USER_OBJ%';
```

Sinonimele dictionarului de date

- Unele dintre vederile dictionarului pot fi apelate si dupa sinonime, de exemplu:

View	Sinonim
DICTIONARY	Dict
USER_OBJECTS	Obj
USER_CATALOG	Cat
USER_TABLES	Tabs
USER_TAB_COLUMNS	Cols
USER_SEQUENCES	Seq
USER_SYNONYM	Syn
USER_INDEXES	Ind

Sinonimele dictionarului de date

- Sinonimele pot fi folosite in interogari. De exemplu, urmatoare cereri sunt echivalente:

SQL> SELECT table_name FROM tabs;

SQL> SELECT table_name FROM user_tables;

- Unele tabele ale sistemului de gestiune contin informatii despre dictionar, de exemplu:
- **DICT_COLUMNS** - contine descriptorul de coloane ale tabelelor si vederilor dictionarului, accesibile userului curent.
- Urmatoarea cerere afiseaza toate coloanele tabelei de dictionar USER_TABLES;

SQL> SELECT table_name, column_name FROM dict_columns
WHERE table_name LIKE 'USER_TABLES';

- Aceleasi informatii se obtin si cu comenzile SQL:

SQL> DESCRIBE user_tables

SQL> DESCRIBE tabs

- De retinut ca sinonimul DICT poate fi folosit cu referire la DICTIONARY:

SQL> DESCRIBE dict

Vederile dictionarului de date

- Vederile dictionarului sunt clasificate in trei grupe, distingandu-se intre ele prin prefixele USER, ALL si DBA care permit userului curent:
  - **USER_xxx** - sa vada informatii despre obiectele create de el si privilegiile acordate pe aceste obiecte.
  - **ALL_xxx** – sa vada obiectele pentru care are drepturile de acces, chiar daca obiectele nu au fost create de el.
  - **DBA_xxx** – sa vada toate obiectele din baza de date care pot fi accesate, daca userul are aprivilegiul DBA.

Vederile dictionarului de date

View	Descriere
USER_CATALOG	Tabele, vederi, sinonime si secvente create de user crt.
USER_CONSTRAINTS	Definitiile constrangerilor pe tabelele userului.
USER_INDEXES	Descrierea indecsilor creati de userul curent.
USER_OBJECTS	Obiecte create de user.
USER_TABLES	Descrierea tabelelor create de userul curent.
USER_TABLESPACES	Descrierea tablespace-urilor accesibile userului.
USER_VIEWS	Descrierea vederilor create de userul curent.

Vederile dictionarului de date

View	Descriere
ALL_CATALOG	Toate tabelele, vederile, sinonimele si secventele accesibile userului.
ALL_COL_COMMENTS	Comentarii pe coloanele tabelelor si vederile accesibile.
ALL_CONSTRAINTS	Definitii de constrangeri pe tabele accesibile.
ALL_CONS_COLUMNS	Informatii despre coloanele accesibile in definitiile de constrangeri.

Vederile dictionarului de date

View	Descriere
ALL_INDEXES	Descrierea indecsilor pe tabelele accesibile userului.
ALL_OBJECTS	Obiectele accesibile userului.
ALL_SYNONYMS	Toate sinonimele accesibile userului.
ALL_TABLES	Descrierea tabelelor accesibile userului.
ALL_TAB_COLUMNS	Coloanele tuturor tabelelor, vederilor si clusterelor.

Vederile dictionarului de date

View	Descriere
ALL_TAB_COMMENTS	Comentarii pe tabele si vederi accesibile userului.
ALL_TAB_PRIVS	Permisiuni pe obiecte pentru care userul este *grantor*,*grantee*,*owner* sau are un rol privat /public.
ALL_USERS	Informatii despre toti userii bazei de date.
ALL_VIEWS	Scriptul de creare a vederilor accesibile userilor.

Vederile dictionarului de date

View	Descriere
DBA_CATALOG	Toate tabelele, vederile, sinonimele si secventele din baza de date.
DBA_COL_COMMENTS	Comentarii pe coloanele tabelelor si vederile accesibile.
DBA_CONSTRAINTS	Constrangeri pe tabelele bazei de date.
DBA_CONS_COLUMNS	Informatii despre tabele si coloanele definite de constrangeri.

Vederile dictionarului de date

View	Descriere
DBA_INDEXES	Descrierea indecsilor creati pe tabelele bazei de date.
DBA_OBJECTS	Toate obiectele bazei de date.
DBA_SYNONYMS	Toate sinonimele create pe baza de date .
DBA_TABLES	Descrierea tabelelor create pe baza de date.
DBA_TAB_COLUMNS	Coloanele tuturor tabelelor, vederilor si clusterelor.

Vederile dictionarului de date

View	Descriere
DBA_TAB_COMMENTS	Coloane si comentarii pe tabele si vederi.
DBA_TAB_PRIVS	Permisiuni pe obiecte pentru care userul este *grantor*,*grantee*,*owner* sau un rol privat /public.
DBA_USERS	Informatii despre toti userii creati pe baza de date.
DBA_VIEWS	Scriptul de creare a tuturor vederilor din baza de date.

Vederile de tip USER

➢ **Vederile de tip USER_xxx**

- Stocheaza informatii despre toate obiectele create de catre userul curent, cum ar fi: tabele, view-uri, indecsi, secvente, functii, proceduri, triggere, etc.
- Vizualizarea acestor tabele se face cu cererea:

```
SQL> SELECT table_name FROM dictionary
        WHERE table_name LIKE 'USER%';
```

- Pentru a vedea ce coloane contine fiecare obiect din dictionar si implicit ce informatii putem sa extragem, folosim urmatoarea comanda:

```
SQL> DESCRIBE user_objects
```

Vederile de tip USER

Exemple:

- Vizualizarea tuturor tabelelor create de catre userul curent:

SQL> SELECT table_name FROM user_tables;

- Vizualizarea tuturor obiectelor create de catre userul curent:

SQL> SELECT object_name, object_type FROM user_objects;

- Vizualizarea tuturor constrangerilor create de catre userul curent:

SQL> SELECT table_name, constraint_name, constraint_type
FROM user_constraints;

- Vizualizarea structurii tabelare a unei tabele:

SQL> SELECT table_name, column_name, data_type
FROM user_tab_columns WHERE table_name='ANGAJATI';

- Vizualizarea tablespace_urilor permanente si temporare alocate userului curent:

SQL> SELECT username, default_tablespace, temporary_tablespace
FROM user_users;

Vederile de tip ALL

➢ **Vederile de tip ALL_xxx**

- Stocheaza informatii despre toate obiectele la care userul curent are acces, atat cele la care este proprietar cat si cele la care a fost grantificat de catre alti useri.
- Vizualizarea acestor vederi se face cu comanda:

SQL> SELECT table_name FROM dictionary

WHERE table_name LIKE 'ALL%';

Exemple:

- Vizualizare obiecte proprii sau create de alti useri la care are acces userul curent:

SQL> SELECT owner,object_name,object_type FROM all_objects

WHERE owner='SCOTT';

- Pentru a vedea scriptul prin care s-a creat un view folosim cererea:

SQL> SELECT view_name, text_vc FROM all_views

WHERE view_name='ANGAJATI_DEP';

Vederile de tip ALL

- Vizualizare ce privilegii sunt acordate de catre un user si pe ce obiecte:

```
SQL> SELECT grantor, grantee, table_name, type, privilege
        FROM all_tab_privs
        WHERE grantor='SCOTT';
```

- Vizualizarea constrangerilor de integritate create de un user si impuse userului curent:

```
SQL> SELECT owner, constraint_name, table_name
        FROM all_constraints WHERE owner='SCOTT';
```

- Vizualizarea indecsilor creati de un user, pe ce tabele si care sunt accesibili userului curent:

```
SQL> SELECT owner, table_name, index_name
        FROM all_indexes WHERE owner='SCOTT';
```

Vederile de tip DBA

➢ **Vederile de tip DBA_xxx**

- Stocheaza informatii despre toate obiectele create pe baza de date, in toate schemele, indiferent de proprietar.
- Pentru a avea acces la aceste informatii userul curent trebuie sa aiba privilegiul DBA.
- Vizualizarea acestor vederi se face cu comanda:

```
SQL> SELECT table_name FROM dictionary
        WHERE table_name LIKE 'DBA%';
```

Exemple:

- Vizualizarea tabelelor, view-urilor si comentariilor create de un user:

```
SQL> COMMENT on table scott.emp_copy is 'tabela copie angajati';
SQL> SELECT table_name, table_type, comments
        FROM dba_tab_comments WHERE owner='SCOTT';
```

Vederile de tip DBA

- Vizualizarea constrangerilor de integritate creati de un user, pe ce tabele au fost create si indecsii creati automat(acolo unde este cazul):

SQL> SELECT constraint_name, constraint_type, table_name, index_name
FROM dba_constraints WHERE owner='SCOTT';

- Vizualizarea constrangerilor de integritate si a coloanelor pe care sunt definite:

SQL> SELECT constraint_name,table_name, column_name
FROM dba_cons_columns WHERE owner='SCOTT';

- Vizualizarea structurii tabelare a unei tabele create de un user:

SQL> SELECT table_name, column_name, data_type
FROM dba_tab_columns
WHERE table_name='ANGAJATI' and owner='SCOTT';

- Informatii despre userii create pe baza de date:

SQL> SELECT username, account_status, created FROM dba_users;

Vederile dinamice

➢ Vederile dinamice

- Sunt vederi ale dictionatului care stocheaza informatii despre obiecte ale bazei de date, in toate schemele, indiferent de proprietar.
- Vederile dinamice au prefixul **V$** , de exemplu:
  - V$DATABASE
  - V$DATAFILE
  - V$CONTROLFILE
  - V$INSTANCE
  - V$SESSION
  - V$PARAMETER
- Pentru a avea acces la aceste informatii userul curent trebuie sa aiba privilegiul DBA. Vizualizarea acestor vederi se face cu comanda:

```
SQL> SELECT table_name FROM dictionary
        WHERE table_name LIKE 'V$%';
```

Vederile dinamice

Exemple:

- Informatii despre baza de date curenta, data cand a fost creata si stare:

```
SQL> SELECT dbid, name, created, log_mode, open_mode
        FROM v$database;
```

- Informatii despre fisierele de date:

```
SQL> SELECT file#, name, block_size, blocks, creation_time, status
        FROM  v$datafile ;
```

- Informatii despre fisierele de control:

```
SQL> select name,  block_size, file_size_blks  FROM v$controlfile;
```

- Informatii despre instanta curenta:

```
SQL> SELECT instance_name, host_name, startup_time, status,
         instance_mode FROM v$instance;
```

- Informatii despre parametrii de sistem:

```
SQL> SELECT name,value FROM v$parameter WHERE name like '%file%';
```