



Sisteme de baze de date

Curs 4 – Normalizarea bazelor de date (cont.)

Limbajul SQL: prelucrarea datelor (1/3)

Sorina Preduț

sorina.predut@unibuc.ro

Universitatea din București



Proiectarea bazelor de date - recap.

- Metodele curente de proiectare a BD sunt divizate în 3 etape:
 1. crearea schemei conceptuale,
 2. crearea design-ului logic al bazei de date și
 3. crearea design-ului fizic al bazei de date.

1. Design conceptual

Diagrama entitate-relație a sistemului prezentat în cursurile anterioare.

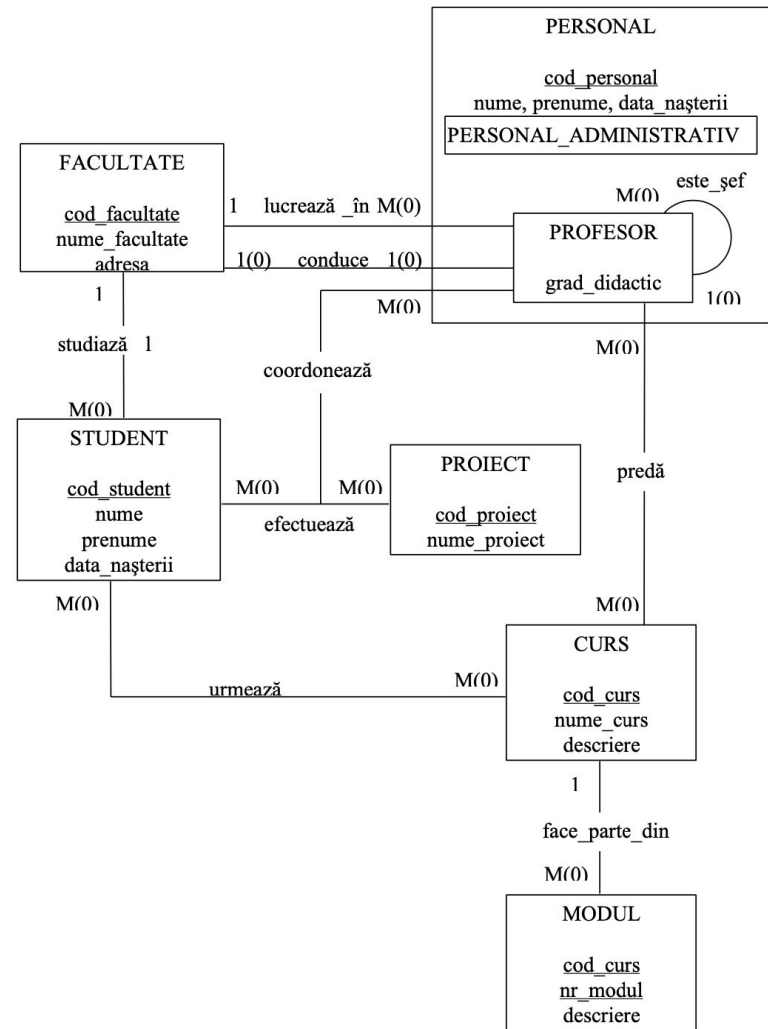
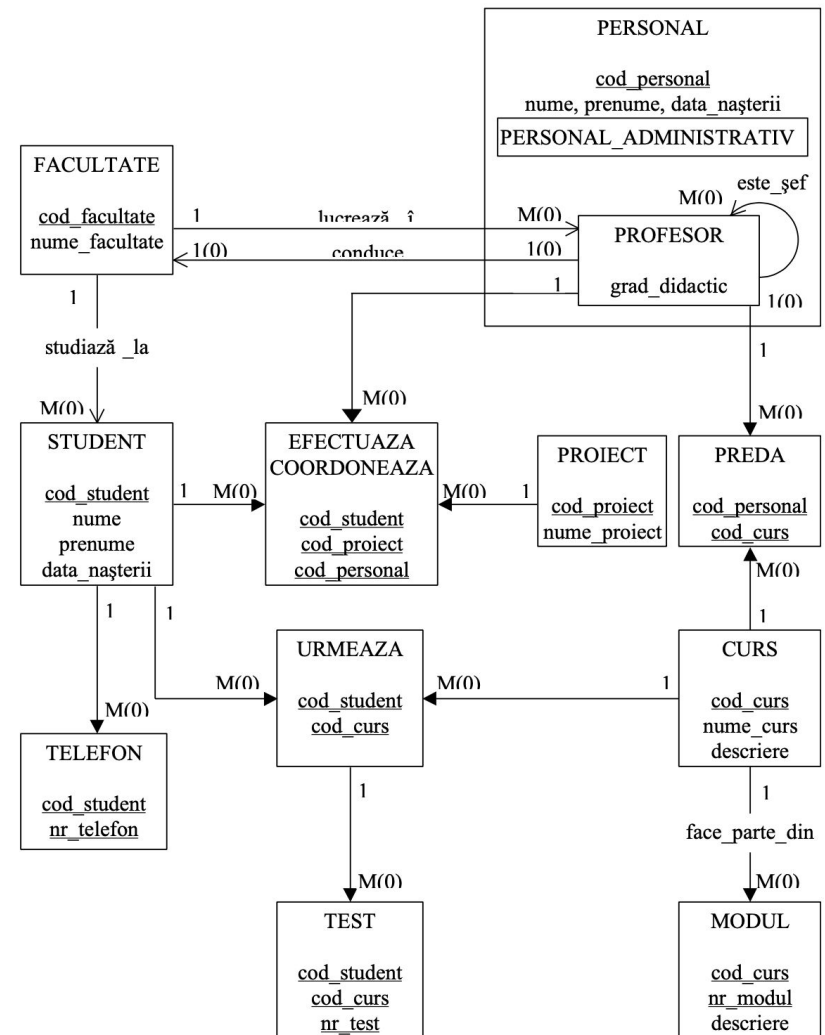


Diagrama conceptuală a sistemului prezentat în cursurile anterioare.





2. Design logic

- Diagrama logică a sistemului prezentat în cursurile anterioare.

PERSONAL (cod_personal, nume, prenume, data_nastere, sex, stare_civila)

PERSONAL_ADMINISTRATIV (cod_personal, profesie, funcție)


PROFESOR (cod_personal, grad_didactic, titlu, sef, ore_predate, data_angajării, *cod_facultate*)

CURS (cod_curs, nume_curs, descriere, nr_ore)

PREDĂ (cod_personal, cod_curs)

MODUL (cod_curs, nr_modul, descriere)

FACULTATE (cod_facultate, nume_facultate, localitate, strada, nr, cod_postal, *cod_decan*)



STUDENT (cod_student, nume, prenume, data_nasterii, tara, localitate, strada, nr, cod_postal, studii_anterioare)

TELEFON (cod_student, nr_telefon, tip_telefon)

PROIECT (cod_proiect, nume_proiect, domeniu)

EFFECTUEAZA_COORDONEAZA (cod_student, cod_proiect, cod_personal)

URMEAZA (cod_student, cod_curs, nota_examen, nota_restantă, observatii)

TEST (cod_student, cod_curs, nr_test, nota_test, observatii)

- **Notă:** Atributele subliniate constituie CP a tabelului, iar cele italice constituie chei străine.

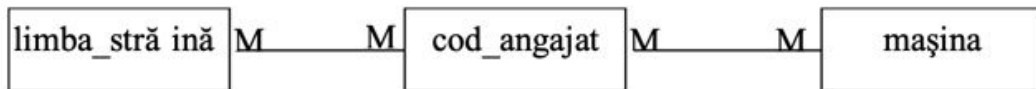


3. Design fizic

- aka Normalizarea bazei de date.

A patra formă normală (4NF)

- Dacă BCNF elimină redundanțele datorate dependențelor funcționale, **4NF determină redundanțele datorate dependențelor multivaloare**.
- Pentru a ilustra acest tip de redundanțe, să considerăm tabelul ANGAJAȚI (cod_angajat, limba_străină, mașina) aflat în BCNF.
- Un angajat poate cunoaște mai multe limbi străine și poate avea mai multe mașini, dar \nexists nici o legătură între limba străină și mașină. Cu alte cuvinte, redundanța datelor din tabelul ANGAJAȚI este cauzată de existența a două relații N:M independente. 4NF va înlătura aceste relații N:M independente.





A patra formă normală (4NF) - cont.

- Fie R un tabel relațional, X și Y două submulțimi de coloane ale lui R și $Z = R - X - Y$ mulțimea coloanelor din R care nu sunt nici în X nici în Y .
Spunem că \exists o **dependență multivaloare** Y de X sau că X **determină multivaloare** pe Y , și notăm $X \twoheadrightarrow Y$, dacă, \forall valoare a coloanelor lui X , sunt asociate valori pentru coloanele din Y care nu sunt corelate în nici un fel cu valorile coloanelor lui Z .

A patra formă normală (4NF) - cont.

- Cu alte cuvinte $X \twoheadrightarrow Y \Leftrightarrow \forall u \text{ și } v, 2 \text{ rânduri ale lui } R \text{ cu}$
 $u(X) = v(X), \exists s \text{ și } t, 2 \text{ rânduri ale lui } R \text{ a.î.}$
 $s(X) = u(X), s(Y) = u(Y), s(Z) = v(Z) \text{ și a.î.}$
 $t(X) = u(X), t(Y) = v(Y), t(Z) = u(Z), \text{ unde prin } u(X) \text{ am notat}$
valoarea coloanelor X corespunzătoare rândului u , etc.
În mod evident, dacă $X \twoheadrightarrow Y$ atunci și $X \twoheadrightarrow Z$.
- Dependența multivaloare se mai numește și **multidependență**.

	X	Y	Z
u	x	y1	z1
v	x	y2	z2
s	x	y1	z2
t	x	y2	z1



A patra formă normală (4NF) - cont.

- În tabelul ANGAJAȚI avem dependențele multivaloare
 $\text{cod_angajat} \twoheadrightarrow \text{limba_străină}$ și
 $\text{cod_angajat} \twoheadrightarrow \text{mașina}$.
Pe de altă parte însă, nici „limba_străină” și nici „mașina” nu depinde funcțional de „cod_angajat”.



A patra formă normală (4NF) - cont.

- Un **tabel relațional R** este în a patra formă normală (4NF) dacă și numai dacă:
 - R este în BCNF.
 - \forall dependență multivaloare $X \twoheadrightarrow Y$ este de fapt o dependență funcțională $X \rightarrow Y$.
- Condiția a doua arată că dacă \exists o dependență multivaloare $X \twoheadrightarrow Y$, atunci \forall coloană, A a lui R va depinde funcțional de coloanele din X, $X \rightarrow A$ - aceasta deoarece existența unei dependențe multivaloare $X \twoheadrightarrow Y$ implică și existența unei dependențe multivaloare $X \twoheadrightarrow (R - X - Y)$.
Ținând cont de faptul că R este în BCNF, înseamnă că \exists o cheie candidată a lui R inclusă în X.
- Un **tabel relațional R** este în a patra formă normală (4NF) \Leftrightarrow
 \forall dependență multivaloare $X \twoheadrightarrow Y \exists$ o cheie a lui R inclusă în X.



Regulă de descompunere

- Un tabel poate fi adus în 4NF prin descompunere fără pierdere de informație astfel:
Fie $R(X, Y, Z)$ un tabel relațional în care \exists o **dependență multivaloare** $X \twoheadrightarrow Y$ astfel încât X nu conține nici o cheie a lui R .
Atunci tabelul R poate fi descompus prin proiecție în două tabele $R1(X, Y)$ și $R2(X, Z)$.
- Aplicând această regulă tabelul ANGAJAȚI se descompune în tabelele
ANGAJAȚI_4A (cod_angajat, limba_străină) și
ANGAJAȚI_4B (cod_angajat, mașina).



Algoritmul 4NFA

1. Se identifică **dependențele multivaloare** $X \twoheadrightarrow Y$ pentru care X și Y nu conțin toate coloanele lui R și X nu conține nici o cheie a lui R .
Se poate presupune că X și Y sunt disjuncte datorită faptului că din $X \twoheadrightarrow Y$ rezultă $X \twoheadrightarrow (Y - X)$.
2. Se înlocuiește tabelul inițial R cu două tabele, primul format din coloanele $\{X, Y\}$, iar celălalt din toate coloanele inițiale, mai puțin coloanele Y .
3. Dacă tabelele rezultate conțin alte dependențe multivaloare, atunci se face transfer la pasul 1, altfel algoritmul se termină.



A cincea formă normală (5NF)

- A cincea formă se întâlnește destul de rar în practică, ea având mai mult valoare teoretică. Dacă în a patra formă normală sunt eliminate relațiile N:M independente, a cincea formă normală are ca scop **eliminarea relațiilor N:M dependente**.

Redundanțele datorate unor astfel de relații pot fi înlăturate prin descompunerea tabelului în 3 sau mai multe tabele.



Exemplu

- Considerăm tabelul LUCRĂTOR_ATELIER_PRODUS (cod lucrător, cod atelier, cod produs), aflat în 4NF. Aparent acest tabel este ilustrarea unei relații de tip 3 care există între lucrător, atelier și produs. Dacă însă presupunem că între lucrător și atelier, lucrător și produs, atelier și produs există relații N:M, atunci în tabel pot exista redundanțe în date care pot fi înlăturate prin descompunerea tabelului LUCRĂTORI în 3 tabele: LUCRĂTOR_ATELIER (cod lucrător, cod atelier), LUCRĂTOR_PRODUS (cod lucrător, cod produs) și ATELIER_PRODUS (cod atelier, cod produs).

LUCRĂTOR_ATELIER_PRODUS

Cod_lucrător	cod_atelier	cod_produs
L1	A1	P1
L2	A1	P2
L2	A1	P1
L2	A2	P1

LUCRĂTOR_ATELIER

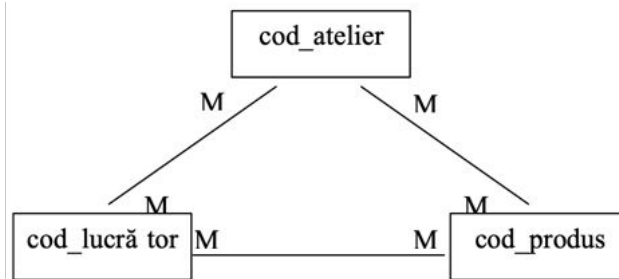
Cod_lucrător	cod_atelier
L1	A1
L2	A1
L2	A2

LUCRĂTOR_PRODUS

Cod_lucrător	cod_produs
L1	P1
L2	P2
L2	P1

ATELIER_PRODUS

Cod_atelier	cod_atelier
A1	P1
A1	P2
A2	P1

**R_12**

Cod_lucrător	cod_atelier	cod_produs
L1	A1	P1
L2	A1	P2
L2	A1	P1
L2	A2	P2
L2	A2	P1

R_13

Cod_lucrător	cod_atelier	cod_produs
L1	A1	P1
L1	A1	P2
L2	A1	P2
L2	A1	P1
L2	A2	P1

R_23

Cod_lucrător	cod_atelier	cod_produs
L1	A1	P1
L1	A2	P1
L2	A1	P2
L2	A1	P1
L2	A2	P1



Exemplu - cont.

- Tabelul LUCRĂTOR_ATELIER elimină redundanța (L2, A1), tabelul LUCRĂTOR_PRODUS elimină redundanța (L2, P1), în timp ce tabelul ATELIER_PRODUS elimină redundanța (A1, P1).
- Tabelul inițial LUCRĂTOR_ATELIER_PRODUS nu poate fi reconstituit din compunerea a doar două din tabelele componente, unde tabelul **R_12** rezultă din compunerea LUCRĂTOR_ATELIER și LUCRĂTOR_PRODUS, **R_13** rezultă din compunerea LUCRĂTOR_ATELIER și ATELIER_PRODUS, iar **R_23** rezultă din compunerea LUCRĂTOR_PRODUS și ATELIER_PRODUS.
- Tabelul inițial LUCRĂTOR_ATELIER_PRODUS poate fi obținut prin compunerea tuturor celor trei tabele componente, de exemplu el rezultă prin compunerea lui R_12 cu ATELIER_PRODUS.



Join-dependență

- Dependența funcțională și dependența multivaloare, și implicit regulile de descompunere pentru formele normale 1NF-4NF, permit **descompunerea prin proiecție a unui tabel relațional în 2 tabele relaționale**.

Totuși, regulile de descompunere asociate acestor forme normale nu dau toate descompunerile posibile prin proiecție a unui tabel relațional.

\exists tabele care nu pot fi descompuse în 2 tabele, dar pot fi descompuse în 3 sau mai multe tabele fără pierdere de informație.

Astfel de descompuneri, în 3 sau mai multe tabele, sunt tratate de 5NF.

Pentru a arăta că un tabel se poate descompune fără pierderi de informație a fost introdus conceptul de **join-dependență** sau **dependență la compunere**.



Join-dependență - cont.

- Fie R un tabel relațional și R_1, R_2, \dots, R_n o mulțime de tabele relaționale care nu sunt disjuncte - au coloane comune - a.î. U coloanelor din R_1, R_2, \dots, R_n este mulțimea coloanelor din R .
Se spune că R satisface **join-dependența** $\{R_1, R_2, \dots, R_n\}$ dacă R se descompune prin proiecție pe R_1, R_2, \dots, R_n fără pierdere de informație, adică tabelul inițial poate fi reconstruit prin compunerea naturală pe attribute comune ale tabelelor rezultate.
- În exemplul anterior, tabelul LUCRĂTOR_ATELIER_PRODUS satisface dependența de uniune $\{LUCRĂTOR_ATELIER, LUCRĂTOR_PRODUS, ATELIER_PRODUS\}$.



Join-dependență - cont.

- Join-dependența este o generalizare a dependenței multivaloare.
Mai precis, **dependența multivaloare corespunde join-dependenței cu două elemente.**
Într-adevăr, dacă în relația $R(X, Y, Z)$ avem multidependență $X \twoheadrightarrow Y$, atunci avem și join-dependență $\{(X \cup Y), (X \cup Z)\}$.
Invers, dacă avem join-dependență $\{R1, R2\}$, atunci avem și dependență multivaloare $(R1 \cap R2) \twoheadrightarrow (R1 - (R1 \cap R2))$.



A cincea formă normală (5NF) - cont.

- Un tabel relațional R este în a cincea formă normală (5NF) dacă și numai dacă \forall join-dependență $\{R_1, R_2, \dots, R_n\}$ este consecința cheilor candidate ale lui R , adică fiecare dintre R_1, R_2, \dots, R_n include o cheie candidată a lui R .



A cincea formă normală (5NF) - cont.

- \forall tabel relațional care este în 5NF este în 4NF deoarece \forall dependență multivaloare poate fi privită ca un caz particular de dependență la uniune.
Trecerea de la 4NF la 5NF constă în identificarea join-dependențelor cu mai mult de 3 elemente și descompunerea tabelului inițial prin proiecție pe aceste componente.
5NF are o importanță practică redusă, cazurile când apare în practică fiind extrem de rare.
- \forall tabel relațional poate fi descompus fără pierderi de informație într-o mulțime de tabele relaționale care sunt în 5NF.
Un tabel în 5NF nu conține anomalii ce pot fi eliminate luând proiecțiile pe diferite submulțimi de coloane ale acestuia.



Concluzii

- Normalizarea este procesul de transformare a structurilor de date și are ca scop **eliminarea redundanțelor și promovarea integrității datelor**.
Normalizarea este un pilon de bază al BD relaționale.
În general, un set de structuri de date nu sunt considerate relaționale decât dacă sunt complet normalizate.
- **Normalizarea** datelor este împărțită în **6** etape, numite **forme normale (NF)**.
Fiecare NF are asociat atât un criteriu cât și un proces.
Criteriul unei anumite NF este mai restrictiv decât al uneia inferioare, a. î.
 \forall tabel relațional care este într-o anumită NF este și în NF inferioară.
Procesul asociat unei NF se referă la trecerea unor structuri de date din NF inferioară în forma normală curentă.



Concluzii - cont.

- $1NF \rightarrow 2NF$ elimină dependențele funcționale parțiale față de chei.
- $2NF \rightarrow 3NF$ elimină dependențele funcționale tranzitive față de chei.
- $3NF \rightarrow BCNF$ elimină dependențele funcționale pentru care determinantul nu este cheie.
- $BCNF \rightarrow 4NF$ elimină toate dependențele multivaloare care nu sunt și dependențe funcționale.
- $4NF \rightarrow 5NF$ elimină toate join-depenențele care nu sunt implicate de o cheie.



Concluzii - cont.

- În practică, cel mai adesea apar primele forme normale (1NF - 3NF), a. î. un tabel relațional aflat în 3NF este de obicei complet normalizat.

În special 5NF apare extrem de rar, având valoare practică foarte scăzută.



Denormalizare

- În principiu, denormalizarea este exact procesul invers normalizării.
- Este procesul de creștere a redundanței datelor, care are ca scop creșterea performanței sau simplificarea programelor de manipulare a datelor.

Totuși, trebuie reținut că o denormalizare corectă **nu înseamnă nicidecum a nu normaliza structurile de date inițiale.**

Din contră, denormalizarea **are loc după ce structurile bazei de date au fost complet normalizate**, și se face prin **selectarea strategică a acelor structuri unde denormalizarea aduce avantaje semnificative.**

Pe de altă parte, orice denormalizare trebuie însoțită de introducerea de măsuri suplimentare, care să asigure integritatea datelor chiar și în cazul unei BD care nu este complet nenormalizată.



Creșterea performanței

- Principalul obiectiv al denormalizării este creșterea performanței programelor de manipulare a datelor.
Una dintre cele mai întâlnite situații de acest gen este denormalizarea folosită pentru **operații sau calcule efectuate frecvent**.
- Să presupunem că pentru înregistrarea tranzacțiilor unui magazin care vinde articole la comandă se folosesc tabelele
VÂNZĂRI_2 (cod comandă, cod articol, cantitate),
ARTICOL (cod articol, nume_articol, cost_articol),
COMANDA_3 (cod comandă, data, cod_client),
CLIENT (cod client, nume_client, nr_telefon)



Creșterea performanței - cont.

- Aceste tabele sunt în 5NF și conțin toate datele necesare pentru obținerea oricărui raport privind vânzările din magazin - de exemplu, în funcție de articol, dată, cantitate, client, etc. Totuși, să presupunem că majoritatea rapoartelor cerute de conducerea magazinului utilizează **cantitatea totală vândută într-o lună pentru fiecare articol**. Tabelele de mai înainte conțin toate datele necesare pentru obținerea acestei informații, de exemplu ea se poate obține printr-o simplă interogare SQL. Pe de altă parte însă, aceasta ar însemna ca **totalul pentru fiecare articol să fie recalculat de fiecare dată când este cerut**.



Creșterea performanței - cont.

- Deoarece este puțin probabil ca acest total să se schimbe după încheierea lunii respective, repetarea acestora ar putea fi înlocuită cu un tabel suplimentar
ARTICOL_LUNA (cod_articol, luna, cantitate_totală)
Datele conținute în acest nou tabel sunt **redundante**, ele putând fi obținute în orice moment din datele conținute în tabelele VÂNZĂRI_2 și COMANDA_3, dar prezintă avantajul că **folosirea lor este mai facilă și mai rapidă decât repetarea calculelor**.
- Pe de altă parte însă, folosirea acestui tabel suplimentar are **și dezavantaje**, putând duce la **pierderea integrității datelor**.
Modificarea datelor din tabelele VÂNZĂRI_2 și COMANDA_3 nu se reflectă automat în datele din noul tabel.



Creșterea performanței - cont.

- De exemplu, dacă după efectuarea calculelor și inserarea totalului în noul tabel se efectuează o vânzare suplimentară, această vânzare nu este reflectată în datele din noul tabel.

De aceea, în situațiile în care BD nu este total normalizată, trebuie luate măsuri suplimentare pentru păstrarea integrității datelor.



Creșterea performanței - cont.

- De exemplu, pentru ca orice modificare a tabelelor inițiale să fie reflectată automat în noul tabel, în Oracle se pot crea așa numitele **triggere** (declanșatoare) ale bazei de date bazate pe tabelele VÂNZĂRI_2 și COMANDA_3 care se declanșează după fiecare actualizare (INSERT, UPDATE, DELETE) a acestor tabele și care actualizează datele din tabelul ARTICOL_LUNA. Aceste triggere pe bază, la rândul lor, vor încetini operațiunile de actualizare din tabelele pe care sunt bazate.



Creșterea performanței - cont.

- Datorită problemelor legate de redundanță și integritate, orice denormalizare trebuie să fie gândită cu atenție pentru a fi siguri că avantajele acesteia în privința performanțelor și a simplității programelor depășesc efortul necesar pentru impunerea integrității.
- În exemplul anterior, se pot crea tabele suplimentare pentru totaluri pe lună sau pe zi. Dacă aceste totaluri sunt utilizate rar, este mai convenabil ca ele să fie calculate direct din tabele normalizate, fără a mai introduce alte date redundante.
- În concluzie, fiecare caz trebuie studiat cu atenție pentru a vedea dacă problemele inerente asociate denormalizării (redundanța, menținerea integrității) sunt compensate de creșterea în performanță pentru anumite situații.



Simplificarea codului

- Un alt motiv invocat pentru folosirea denormalizării este acela al simplificării codului pentru manipularea datelor.
Cu alte cuvinte, pentru un dezvoltator, un singur tabel poate părea uneori mai ușor de utilizat decât două sau mai multe.
De exemplu, să considerăm tabelul
STOCURI(cod_depozit, cod_material, nume_material, cantitate)
utilizat de o firmă pentru a înregistra cantitățile de materiale existente în fiecare din depozitele sale.
Evident, acest tabel nu este în 2NF și poate fi normalizat prin descompunerea în 2 tabele:
STOCURI_2(cod_depozit, cod_material, cantitate)
MATERIAL(cod_material, nume_material)



Simplificarea codului - cont.

- Totuși, dacă se dorește aflarea tuturor depozitelor în care există ciment, de exemplu, pentru dezvoltator este mai convenabil să folosească varianta nenormalizată, în acest caz interogarea SQL corespunzătoare fiind

```
SELECT cod_depozit, cantitate  
FROM stocuri  
WHERE nume_material = 'CIMENT';
```

- Evident, interogarea de mai sus este mai simplă decât varianta în care se folosesc cele două tabele normalizate

```
SELECT cod_depozit, cantitate  
FROM stocuri_2, material  
WHERE stocuri_2.cod_material = material.cod_material AND  
      nume_material = 'CIMENT';
```



Simplificarea codului - cont.

- Pe de altă parte însă, diferența de performanță (timp de execuție) dintre cele două variante de interogări este neglijabilă, de multe ori chiar interogarea pe structuri normalizate va fi mai rapidă decât cealaltă, astfel încât singurul avantaj al folosirii structurii nenormalizate este simplitatea.
- Dacă pentru un dezvoltator simplificarea invocată mai sus este nesemnificativă - orice dezvoltator trebuie să fie capabil să scrie o interogare pe mai multe tabele de tipul celei de mai sus - problema se pune cu mai mare stringență în cazul în care utilizatorii pot să-și scrie propriile interogări ale bazei de date.



Simplificarea codului - cont.

- În acest caz, pentru a păstra atât avantajele normalizării și pentru a permite în același timp simplificarea interogărilor, se poate crea o vedere bazată pe cele două tabele normalizate

```
CREATE VIEW stocuri AS
SELECT cod_depozit, stocuri_2.cod_material, nume_material, cantitate
FROM stocuri_2, material
WHERE stocuri_2.cod_material = material.cod_material;
```
- O astfel de vedere „ascunde” tabelele normalizate și permite utilizatorilor scrierea unor interogări mai simple.

Pe de altă parte însă, performanța programelor poate scădea mult în cazul vederilor complexe, în acest caz fiind uneori de preferat a se utiliza tabelele de bază pentru creșterea performanței.



Concluzii

- Designul logic al BD are un **impact** profund asupra **performanțelor și structurii sistemului**. Deși este posibil a compensa un design prost prin scrierea unui volum mare de cod, acest lucru nu este în nici un caz de dorit, mai ales că în majoritatea cazurilor el va duce inevitabil la scăderea performanțelor sistemului.
- În mod ideal, dezvoltarea oricărei aplicații trebuie să înceapă de la o structură complet normalizată, care să constituie o temelie solidă a acesteia.
- Denormalizarea, dacă ea este necesară, trebuie făcută numai după ce se obține o structură complet normalizată.
Ca regulă generală, denormalizarea este folosită rar și numai după o analiză atentă.



Limbajul SQL: prelucrarea datelor (1/3)



Comanda SELECT

- Regăsirea datelor stocate în BD este considerată cea mai importantă facilitare a unui SGBD. În SQL ea se realizează prin intermediul comenzii SELECT. Aceasta este folosită doar pentru regăsirea datelor, ea neputându-le modifica.
- Comanda SELECT implementează toți operatorii algebrei relaționale.
- O instrucțiune SELECT cuprinde în mod obligatoriu cuvântul cheie FROM.
Cu alte cuvinte, sintaxa minimală pentru comanda SELECT este:
`SELECT attribute FROM tabel;`
- După cuvântul cheie SELECT se specifică lista atributelor ce urmează a fi returnate ca rezultat al interogării, iar după cuvântul FROM se precizează obiectele (tabele, vederi, sinonime) din care se vor selecta aceste attribute.

Atributele comenzii SELECT

- În lista de atribute pot apărea:
 - toate coloanele din tabel sau vedere (în ordinea în care au fost definite în comanda CREATE TABLE / CREATE VIEW prin utilizarea semnului * :

```
SELECT * FROM profesor;
```

COD	NUME	PRENUME	DATA_NAST	GRAD	SEF	SALARIU	PRIMA	COD_CATEDRA
----	-----	-----	-----	-----	---	-----	-----	-----
100	GHEORGHIU	STEFAN	11-AUG-46	PROF	3000	3500		10
101	MARIN	VLAD	19-APR-45	PROF	100	2500		20
102	GEORGESCU	CRISTIANA	30-OCT-51	CONF	100	2800	200	30
103	IONESCU	VERONICA		ASIST	102	1500		10
104	ALBU	GHEORGHE		LECT	100	2200	2500	20
105	VOINEA	MIRCEA	15-NOV-65	ASIST	100	1200	150	10
106	STANESCU	MARIA	05-DEC-69	ASIST	103	1200	600	20



Atributele comenzii SELECT - cont.

- numele coloanelor separate prin virgulă. Acestea vor apărea în rezultatul interogării în ordinea în care sunt specificate:

```
SELECT nume, prenume, salariu FROM profesor;
```

NUME	PRENUME	SALARIU
-----	-----	-----
GHEORGHIU	STEFAN	3000
MARIN	VLAD	2500
GEORGESCU	CRISTIANA	2800
IONESCU	VERONICA	1500
ALBU	GHEORGHE	2200
VOINEA	MIRCEA	1200
STANESCU	MARIA	1200



Atributele comenzii SELECT - cont.

- atribute rezultate din evaluarea unor expresii. Aceste expresii pot conține nume de coloane, constante, operatori sau funcții.



Operatori aritmetici

NUMES	SALARIU	SALARIU*0.38
-----	-----	-----
GHEORGHIU	3000	1140
MARIN	2500	950
GEORGESCU	2800	1064
IONESCU	1500	570
ALBU	2200	836
VOINEA	1200	456
STANESCU	1200	456

- Operatorii aritmetici pot fi folosiți pentru a crea expresii având tipul de date numeric sau date calendaristice.

Operatorii aritmetici sunt: + (adunare), - (scădere), * (înmulțire), / (împărțire).

Ordinea de precedență a operatorilor poate fi schimbată cu ajutorul parantezelor.

- De exemplu dacă în tabela PROFESOR ne interesează să calculăm impozitul aferent salariilor, știind că acesta este de 38%, putem scrie următoarea interogare:

```
SELECT nume, salariu, salariu*0.38  
FROM profesor;
```



Aliasuri de coloane

- Pentru exemplul anterior observăm că în momentul afișării rezultatelor, SQL*Plus utilizează numele coloanelor ca antet.
Când acest lucru poate face dificilă înțelegerea rezultatelor, se poate schimba antetul prin atribuirea altor nume coloanelor (numite 'alias'-uri ale coloanei).
Acest lucru se realizează specificând aliasul după numele coloanei.
În cazul în care aliasul conține spații sau caractere speciale cum ar fi + sau -, acesta se va specifica între ghilimele.
- În exemplul următor aliasul "DATA NASTERE" conține spații deci este specificat între ghilimele în timp ce aliasul IMPOZIT nu conține spații sau caractere speciale deci nu trebuie specificat obligatoriu între ghilimele.



Aliasuri de coloane - cont.

```
SELECT nume, data_nast "DATA NASTERE", salariu,  
       salariu*0.38 IMPOZIT  
FROM profesor;
```

NUME	DATA NASTERE	SALARIU	IMPOZIT
GHEORGHIU	11-AUG-46	3000	1140
MARIN	19-APR-45	2500	950
GEORGESCU	30-OCT-51	2800	1064
IONESCU		1500	570
ALBU		2200	836
VOINEA	15-NOV-65	1200	456
STANESCU	05-DEC-69	1200	456



Operatorul de concatenare

DETALII_PROFESOR

100 GHEORGHIU STEFAN
101 MARIN VLAD
102 GEORGESCU CRISTIANA
103 IONESCU VERONICA
104 ALBU GHEORGHE
105 VOINEA MIRCEA
106 STANESCU MARIA

- Operatorul de concatenare, notat ||, permite legarea coloanelor cu alte coloane, expresii aritmetice sau valori constante pentru a crea o expresie de tip șir de caractere.
- De exemplu, pentru a combina codul, numele și prenumele unui profesor, separate printr-un spațiu, se folosește următoarea interogare:

```
SELECT cod || ' ' || nume || ' '  
        || prenume detalii_profesor  
FROM profesor;
```



Convertirea valorilor Null cu ajutorul funcției NVL

NUME	SALARIU	PRIMA	SALARIU TOTAL
GHEORGHIU	3000	3500	6500
MARIN	2500		
GEORGESCU	2800	200	3000
IONESCU	1500		
ALBU	2200	2500	4700
VOINEA	1200	150	1350
STANESCU	1200	600	1800

- Dacă la o înregistrare pentru o anumită coloană valoarea este necunoscută sau neaplicabilă, atunci aceasta este Null. Această valoare nu trebuie confundată cu zero sau șirul de caractere format dintr-un spațiu. Așa cum am văzut în exemplele de până acum, dacă o anumită valoare este Null, SQL*Plus nu va afișa nimic.
Pentru expresiile aritmetice, dacă una dintre valorile componente este Null, atunci și rezultatul expresiei este Null.
- De exemplu, pentru a calcula salariul total, ce reprezintă suma dintre coloanele salariu și prima putem folosi interogarea:

```
SELECT nume, salariu, prima,  
       salariu+prima "SALARIU TOTAL"  
FROM profesor;
```




Convertirea valorilor Null - cont.

- Observăm că pentru acele înregistrări care au avut valoarea Null în câmpul `prima` expresia ce calculează salariul total returnează tot valoarea Null.
- Pentru a obține un rezultat diferit de Null, **valorile Null trebuie convertite într-un număr (în cazul de față 0) înainte de a aplica operatorul aritmetic.**

Această convertire se poate realiza prin intermediul funcției NVL.

Funcția **NVL** are 2 argumente.

Dacă valoarea primului argument nu este Null, atunci NVL întoarce această valoare; dacă nu, ea întoarce valoarea celui de-al doilea argument.

Cele 2 argumente pot avea orice tip de date. Dacă tipurile de date ale celor 2 argumente diferă, Oracle încercă să convertească al doilea argument la tipul de date al primului.

Convertirea valorilor Null - cont.

- De exemplu, pentru a putea calcula salariul total al tuturor cadrelor didactice, trebuie să convertim valoarea Null din coloana prima a tabelului PROFESOR în valoarea 0 folosind NVL(prima, 0):

```
SELECT nume, salariu, prima, salariu+NVL(prima,0) "SALARIU TOTAL"
```

```
FROM profesor;
```

NUME	SALARIU	PRIMA	SALARIU TOTAL
------	---------	-------	---------------

GHEORGHIU	3000	3500	6500
MARIN	2500		2500
GEORGESCU	2800	200	3000
IONESCU	1500		1500
ALBU	2200	2500	4700
VOINEA	1200	150	1350
STANESCU	1200	600	1800



Prevenirea selectării înregistrărilor duplicat

- O comanda SELECT care nu cuprinde cuvântul cheie DISTINCT va afișa toate înregistrările care rezultă din interogare, indiferent dacă una sau mai multe dintre ele sunt identice.

- De exemplu, interogarea de mai jos va returna următoarele rezultate:

```
SELECT grad FROM profesor;
```

- În cazul folosirii cuvântului cheie DISTINCT înregistrările duplicat sunt eliminate, afișându-se numai prima apariție a valorilor câmpurilor specificate în lista de attribute.

- De exemplu:

```
SELECT DISTINCT grad FROM profesor;
```

GRAD

PROF
PROF
CONF
ASIST
LECT
ASIST
ASIST

GRAD

ASIST
CONF
LECT
PROF



Prevenirea selectării înreg. duplicat - cont.

- Dacă lista de attribute conține mai multe coloane, operatorul DISTINCT va afecta toate coloanele selectate.
- Următorul exemplu va afișa toate combinațiile de valori care sunt diferite pentru coloanele grad și cod_catedra.

Exemplu:

```
SELECT DISTINCT grad, cod_catedra  
FROM profesor;
```

GRAD	COD_CATEDRA
-----	-----
ASIST	10
ASIST	20
CONF	30
LECT	20
PROF	10
PROF	20



Clauza ORDER BY

- În mod normal, în urma interogării înregistrările rezultate apar în aceeași ordine în care au fost introduse în baza de date.
- Pentru a modifica ordinea de afișare se utilizează clauza ORDER BY, care **sortează înregistrările după valorile din una sau mai multe coloane**.
- Această clauză este urmată de numele coloanelor după care se va face sortarea. De asemenea, este posibil să se identifice coloana dintr-o clauză ORDER BY folosind în locul numelui coloanei un număr ordinal ce reprezintă poziția coloanei în rezultat (de la stânga la dreapta).
- Această facilitate face posibilă ordonarea rezultatului interogării în funcție de un atribut al clauzei SELECT care poate fi o expresie complexă, fără a mai rescrie acea expresie.



Clauza ORDER BY - cont.

- Nu există nici o limită a nr. de coloane în funcție de care se poate face sortarea.
- Nu este obligatoriu ca ordinea de sortare să se facă în funcție de o coloană care să fie afișată, dar în acest caz nu se mai poate folosi numărul de ordine al coloanei în loc de numele acesteia.
- Înregistrările vor fi sortate mai întâi în funcție de primul câmp specificat după clauza ORDER BY, apoi, înregistrările care au aceeași valoare în acest prim câmp sunt sortate în funcție de valoarea celui de-al doilea câmp specificat după clauza ORDER BY, ș.a.m.d.

Clauza ORDER BY - cont.

- De exemplu, pentru a sorta ascendent înregistrările în funcție de impozitul pe salariu folosim interogarea:

```
SELECT nume, salariu*0.38  
FROM profesor  
ORDER BY salariu*0.38;
```

care este echivalentă cu:

```
SELECT nume, salariu*0.38  
FROM profesor  
ORDER BY 2;
```

NUME	SALARIU*0.38
-----	-----
VOINEA	456
STANESCU	456
IONESCU	570
ALBU	836
MARIN	950
GEORGESCU	1064
GHEORGHIU	1140

Clauza ORDER BY - cont.

- Înregistrările sunt sortate în mod implicit în ordine ascendentă (opțiunea ASC), afișarea în ordine descendentă făcându-se prin utilizarea opțiunii DESC.
- Observați că în momentul sortării **valoarea Null este considerată cea mai mare, deci dacă sortarea este ascendentă este trecută pe ultima poziție și dacă sortarea este descendentă este trecută pe prima poziție.**
- De exemplu:

```
SELECT grad, prima  
FROM profesor  
ORDER BY grad, prima DESC;
```

GRAD	PRIMA
ASIST	
ASIST	600
ASIST	150
CONF	200
LECT	2500
PROF	
PROF	3500



Clauza ORDER BY - cont.

- Se observă că în exemplul anterior înregistrările au fost mai întâi sortate ascendent (specificație implicită) în funcție de gradul didactic.
Înregistrările cu același grad au fost apoi ordonate în funcție de cel de-al doilea criteriu de sortare, adică în funcție de prima primită cu specificația explicită de sortare descendentă.



Clauza WHERE

- Clauza WHERE se folosește **pentru a regăsi înregistrări ce corespund unei anumite condiții evaluate cu valoarea de adevăr TRUE**, adică pentru a realiza anumite restricții de selecție.
- Astfel, clauza WHERE corespunde restricțiilor operatorilor din algebra relațională. Cu alte cuvinte, dacă o clauză ORDER BY este o clauză de sortare, **clauza WHERE este o clauză de filtrare**.

Dacă nu se specifică o clauză WHERE, interogarea va întoarce ca rezultat toate rândurile din tabel.

Împreună cu clauza SELECT și FROM care sunt obligatorii, WHERE este cea mai folosită clauză din SQL.



Clauza WHERE - cont.

- Din punct de vedere sintactic clauza WHERE este opțională, dar atunci când este introdusă urmează întotdeauna imediat după clauza FROM:

```
SELECT numecoloană  
FROM tabel  
WHERE condiție;
```



Clauza WHERE - cont.

- Datorită existenței valorii Null, în SQL o condiție poate lua atât valorile TRUE și FALSE cât și valoarea Necunoscut (despre acest lucru vom discuta în detaliu mai târziu).
- O comandă SELECT cu clauza WHERE va returna toate înregistrările pentru care condiția are valoarea TRUE.

Condiția clauzei WHERE poate cuprinde numele unor coloane, constante, operatori de comparație sau operatori logici (NOT, AND, OR).

Operatorii de comparație se pot împărți în 2 categorii: operatori relaționali și operatori SQL. Toți acești operatori sunt trecuți în revistă în continuare.



Operatori relaționali

- Operatorii relaționali sunt:
 - = egal
 - > mai mare
 - >= mai mare sau egal
 - < mai mic
 - <= mai mic sau egal
 - <> și != diferit
- Cele 2 valori care sunt comparate trebuie să aparțină unor tipuri de date compatibile.



Operatori relaționali - cont.

- De exemplu, pentru a selecta toate cadrele didactice care nu aparțin catedrei cu codul 10 folosim următoarea interogare:

```
SELECT nume, prenume  
FROM profesor  
WHERE cod_catedra <>10;
```

NUME	PRENUME
-----	-----
MARIN	VLAD
GEORGESCU	CRISTIANA
ALBU	GHEORGHE
STANESCU	MARIA



Operatori relaționali - cont.

- Șirurile de caractere și data calendaristică trebuie incluse între apostrofuri.
- De exemplu, pentru a selecta numai acele cadre didactice care au gradul didactic de profesor vom utiliza următoarea interogare:

```
SELECT nume, prenume  
FROM profesor  
WHERE grad = 'PROF';
```

NUME	PRENUME
-----	-----
GHEORGHIU	STEFAN
MARIN	VLAD



Operatori relaționali - cont.

- În cazul șirurilor de caractere, literele mici sunt diferite de literele mari.
- De exemplu, următoarea interogare nu va returna nici o înregistrare:

```
SELECT nume, prenume  
FROM profesor  
WHERE grad = 'prof';
```




Operatori relaționali - cont.

- Toți operatorii de comparație pot fi folosiți atât pentru valori numerice cât și pentru șiruri de caractere sau date calendaristice.
- De exemplu, pentru a afla toate cadrele didactice care s-au născut înainte de 1 Ianuarie

1965 folosim interogarea:

```
SELECT nume, prenume, data_nast  
FROM profesor  
WHERE data_nast < '01-JAN-65';
```

NUME	PRENUME	DATA_NAST
-----	-----	-----
GHEORGHIU	STEFAN	11-AUG-46
MARIN	VLAD	19-APR-45
GEORGESCU	CRISTIANA	30-OCT-51

Operatori relaționali - cont.

- În cazul șirurilor de caractere ordonarea se face după codul ASCII al acestora.
- De exemplu, pentru a afla toate cadrele didactice ale căror nume sunt în ordinea alfabetică după litera 'M' se poate folosi interogarea:

```
SELECT nume, prenume  
FROM profesor  
WHERE nume >= 'M';
```

NUME	PRENUME
-----	-----
MARIN	VLAD
VOINEA	MIRCEA
STANESCU	MARIA

- Observați că interogarea de mai sus este corectă numai în cazul în care numele angajaților începe cu o literă mare, literele mici fiind în spatele celor mari.



Operatori relaționali - cont.

- Există posibilitatea de a compara pentru o înregistrare valoarea unei coloane cu valoarea altei coloane.
- De exemplu, dacă dorim să selectăm acele cadre didactice care au primit primă mai mare decât salariul de bază vom avea:

```
SELECT nume, prenume, salariu, prima
```

```
FROM profesor
```

```
WHERE salariu < prima;
```

NUME	PRENUME	SALARIU	PRIMA
-----	-----	-----	-----
GHEORGHIU	STEFAN	3000	3500
ALBU	GHEORGHE	2200	2500



Operatori SQL

- Există 4 tipuri de operatori SQL care pot opera cu toate tipurile de date:
 - BETWEEN ... AND ...
 - IN
 - LIKE
 - IS NULL

Operatorul BETWEEN ... AND ...

- Operatorul BETWEEN ... AND ... permite specificarea unui domeniu mărginit de 2 valori între care trebuie să se afle valoarea testată.
Domeniul de valori specificat este un interval închis iar limita inferioară trebuie specificată prima.
- Astfel, dacă dorim selectarea acelor cadre didactice care au salariul între 2000 și 3000 vom folosi comanda:

```
SELECT nume, prenume, salariu  
FROM profesor  
WHERE salariu BETWEEN 2000 AND 3000;
```

NUME	PRENUME	SALARIU
-----	-----	-----
GHEORGHIU	STEFAN	3000
MARIN	VLAD	2500
GEORGESCU	CRISTIANA	2800
ALBU	GHEORGHE	2200

Operatorul IN

- Operatorul IN permite specificarea unei liste de valori, valoarea testată trebuind să se afle printre valorile acestei liste.
- De exemplu, dacă dorim selectarea cadrelor didactice care au gradul de conferențiar, lector sau asistent vom utiliza comanda:

```
SELECT nume, prenume, grad
```

```
FROM profesor
```

```
WHERE grad IN ('CONF', 'LECT', 'ASIST');
```

NUME	PRENUME	GRAD
-----	-----	----
GEORGESCU	CRISTIANA	CONF
IONESCU	VERONICA	ASIST
ALBU	GHEORGHE	LECT
VOINEA	MIRCEA	ASIST
STANESCU	MARIA	ASIST



Operatorul LIKE

- Operatorul LIKE permite specificarea unui anumit model de șir de caractere cu care trebuie să se potrivească valoarea testată.
- Acest operator se folosește în mod special atunci când nu se știe exact valoarea care trebuie căutată.
- Pentru a construi modelul după care se face căutarea pot fi folosite 2 simboluri:
 - % semnifică orice secvență de zero sau mai multe caractere,
 - _ semnifică orice caracter (care apare o singură dată).

Operatorul LIKE - cont.

- De exemplu, următoarea comandă SELECT va returna toate cadrele didactice al căror nume începe cu litera 'G':

SELECT nume, prenume	NUME	PRENUME
FROM profesor	-----	-----
WHERE nume LIKE 'G%';	GHEORGHIU	STEFAN
	GEORGESCU	CRISTIANA

- Dacă dorim selectarea acelor cadre didactice al căror nume are litera 'O' pe a doua poziție, indiferent de lungimea cuvântului, vom avea:

SELECT nume, prenume	NUME	PRENUME
FROM profesor	-----	-----
WHERE nume LIKE '_O%';	IONESCU	VERONICA
	VOINEA	MIRCEA



Operatorul LIKE - cont.

- O problemă intervine atunci când șirul conține caracterele % sau _ (de exemplu șirul 'J_James') deoarece aceste caractere au semnificație predefinită.
Pentru a schimba interpretarea acestor caractere se folosește opțiunea ESCAPE.
- De exemplu, pentru a căuta toate titlurile de carte care încep cu caracterele 'J_' se poate folosi interogarea:

```
SELECT titlu FROM carte WHERE titlu LIKE 'J/_%' ESCAPE '/';
```


Opțiunea ESCAPE identifică caracterul '/' ca fiind caracterul "escape".
Deoarece în modelul folosit pentru LIKE acest caracter precede caracterul '_' acesta din urmă va fi interpretat ca o simplă literă, fără altă semnificație.
- Avantajul unei viteze mari de regăsire ca urmare a indexării este pierdut în momentul în care se caută un șir de caractere care începe cu "_" sau "%" într-o coloană indexată.



Operatorul IS NULL

- Operatorul IS NULL testează dacă valorile sunt Null.
- Pentru a vedea utilitatea acestui operator să considerăm următoarele interogări:

```
SELECT nume, prenume FROM profesor WHERE prima = NULL;
```

```
SELECT nume, prenume FROM profesor WHERE prima <> NULL;
```

Amândouă aceste interogări nu vor returna nici o înregistrare.

Acest lucru pare surprinzător la prima vedere deoarece ne-am fi așteptat ca prima interogare să returneze toate cadrele didactice care nu au primit primă, iar a doua toate cadrele didactice care au primit primă.

În SQL însă, orice condiție care este formată dintr-un operator de comparație care are unul dintre termeni valoarea Null va avea ca rezultat valoarea Necunoscut, diferită de valoarea TRUE (pentru care se face filtrarea).

Operatorul IS NULL - cont.

- Pentru compararea cu Null se folosește operatorul special IS NULL.
- Deci pentru a afla cadrele didactice care nu au primit primă se folosește interogarea:

SELECT nume, prenume	NUME	PRENUME
FROM profesor	-----	-----
WHERE prima IS NULL;	MARIN	VLAD
	IONESCU	VERONICA

- La fel, pentru a afla cadrele didactice ale căror dată de naștere nu se cunoaște vom folosi următoarea interogare:

SELECT nume, prenume	NUME	PRENUME
FROM profesor	-----	-----
WHERE data_nast IS NULL;	IONESCU	VERONICA
	ALBU	GHEORGHE



Operatori logici

- Există 3 tipuri de operatori logici:
 - NOT
 - AND
 - OR



Negarea operatorilor

- În unele cazuri sunt mai ușor de căutat înregistrările care nu îndeplinesc o anumită condiție. Acest lucru se poate realiza folosind operatorul NOT.
- Operatorul NOT se poate folosi pentru negarea unei expresii logice (de exemplu expresii de tipul NOT coloana = ...) sau pentru negarea operatorilor SQL în modul următor:
 - NOT BETWEEN
 - NOT IN
 - NOT LIKE
 - IS NOT NULL.



Negarea operatorilor - cont.

- De exemplu, pentru a selecta cadrele didactice al căror nume nu începe cu litera 'G' se folosește interogarea:

```
SELECT nume, prenume  
FROM profesor  
WHERE nume NOT LIKE 'G%';
```

NUME	PRENUME
-----	-----
MARIN	VLAD
IONESCU	VERONICA
ALBU	GHEORGHE
VOINEA	MIRCEA
STANESCU	MARIA



Negarea operatorilor - cont.

- Pentru a selecta cadrele didactice care au primă se folosește interogarea:

```
SELECT nume, prenume  
FROM profesor  
WHERE prima IS NOT NULL;
```

NUME	PRENUME
-----	-----
GHEORGHIU	STEFAN
GEORGESCU	CRISTIANA
ALBU	GHEORGHE
VOINEA	MIRCEA
STANESCU	MARIA



Negarea operatorilor - cont.

- **Observație:** Negarea unei expresii logice care are valoarea Necunoscut va avea tot valoarea Necunoscut.
- De exemplu, o expresie de tipul NOT coloana = NULL va avea valoarea Necunoscut, următoarea interogare nereturnând nici o înregistrare:

```
SELECT nume, prenume  
FROM profesor  
WHERE NOT prima = NULL;
```




Condiții multiple de interogare (op. AND și OR)

- Operatorii AND și OR pot fi utilizați pentru a realiza interogări ce conțin condiții multiple. Condiția ce conține operatorul AND este adevărată atunci când ambele condiții sunt adevărate iar condiția ce conține operatorul OR este adevărată atunci când cel puțin una din condiții este adevărată.
- În aceeași expresie logică se pot combina operatorii AND și OR dar operatorul AND are o precedență mai mare decât operatorul OR deci este evaluat mai întâi.
- În momentul evaluării unei expresii, se calculează mai întâi operatorii în ordinea precedenței, de la cel cu precedența cea mai mare până la cel cu precedența cea mai mică. Dacă operatorii au precedență egală atunci ei sunt calculați de la stânga la dreapta.



Condiții multiple de interogare (op. AND și OR)

- Precedența operatorilor, pornind de la cea mai mare la cea mai mică este următoarea:
 - toți operatorii de comparație și operatorii SQL: >, <, <=, >=, =, <>, BETWEEN ... AND ..., IN, LIKE, IS NULL;
 - operatorul NOT;
 - operatorul AND;
 - operatorul OR.
- Pentru a schimba prioritatea operatorilor se folosesc parantezele.
- În exemplele următoare se observă modul de evaluare al expresiei în funcție de precedența operatorilor, precum și modul în care parantezele pot schimba acest lucru.



Condiții multiple de interogare (op. AND și OR)

```
SELECT nume, prenume, salariu, cod_catedra  
FROM profesor  
WHERE salariu>2000 AND cod_catedra=10 OR cod_catedra=20;
```

este echivalentă cu:

```
SELECT nume, prenume, salariu, cod_catedra  
FROM profesor  
WHERE (salariu>2000 AND cod_catedra=10)OR cod_catedra=20;
```

NUME	PRENUME	SALARIU	COD_CATEDRA
-----	-----	-----	-----
GHEORGHIU	STEFAN	3000	10
MARIN	VLAD	2500	20
ALBU	GHEORGHE	2200	20
STANESCU	MARIA	1200	20



Condiții multiple de interogare (op. AND și OR)

```
SELECT nume, prenume, salariu, cod_catedra
FROM profesor
WHERE salariu>2000 AND (cod_catedra=10 OR cod_catedra=20);
```

NUME	PRENUME	SALARIU	COD_CATEDRA
-----	-----	-----	-----
GHEORGHIU	STEFAN	3000	10
MARIN	VLAD	2500	20
ALBU	GHEORGHE	2200	20



Funcții

- Funcțiile sunt o caracteristică importantă a SQL și sunt utilizate pentru a realiza calcule asupra datelor, modifica date, manipula grupuri de înregistrări, schimba formatul datelor sau pentru a converti diferite tipuri de date.

Funcțiile se clasifică în 2 tipuri:

- Funcții referitoare la o singură înregistrare:
 - funcții caracter;
 - funcții numerice;
 - funcții pentru data calendaristică și oră;
 - funcții de conversie;
 - funcții diverse.
- Funcții referitoare la mai multe înregistrări: funcții totalizatoare sau funcții de grup.



Funcții

- Diferența dintre cele 2 tipuri de funcții este nr. de înregistrări pe care acționează. Funcțiile referitoare la o singură înregistrare returnează un singur rezultat pentru fiecare rând al tabelului, pe când funcțiile referitoare la mai multe înregistrări returnează un singur rezultat pentru fiecare grup de înregistrări din tabel.
- O observație importantă este faptul că dacă se apelează o funcție SQL ce are un argument egal cu valoarea Null, atunci în mod automat rezultatul va avea valoarea Null. Singurele funcții care nu respectă această regulă sunt: CONCAT, DECODE, DUMP, NVL și REPLACE.
- În continuare vom exemplifica și prezenta la modul general cele mai importante funcții, pentru sintaxă.



Funcții referitoare la o singură înregistrare

- Sunt funcții utilizate pentru manipularea datelor individuale.
Ele pot avea unul sau mai multe argumente și returnează o valoare pentru fiecare rând rezultat în urma interogării.
- Funcții caracter
 - Aceste funcții au ca argumente date de tip caracter și returnează date de tip VARCHAR2, CHAR sau NUMBER.



Funcții caracter

- Cele mai importante funcții caracter sunt:
 - CONCAT - returnează un șir de caractere format prin concatenarea a două șiruri;
 - LOWER - modifică toate caracterele în litere mici;
 - UPPER - modifică toate caracterele în litere mari;
 - LENGTH - returnează nr. de caractere dintr-un anumit câmp;
 - REPLACE - caută într-un șir de caractere un subșir iar dacă îl găsește îl va înlocui cu un alt șir de caractere;
 - SUBSTR - returnează un subșir de caractere având o anumită lungime începând cu o anumită poziție;
 - TRANSLATE - caută într-un șir de caractere un caracter iar dacă îl găsește îl va înlocui cu un alt caracter;



Funcții caracter - cont.

- Exemplu de utilizare a funcției LENGTH:

```
SELECT LENGTH (nume)  
FROM profesor;
```

LENGTH(NUME)

9
5
9
7
4
6
8



Funcții caracter - cont.

- Spre deosebire de alte funcții, funcțiile caracter pot fi imbricate până la orice adâncime.
- Dacă funcțiile sunt imbricate ele sunt evaluate din interior spre exterior.
- Pentru a determina, de exemplu, de câte ori apare caracterul 'A' în câmpul nume vom folosi interogarea:

```
SELECT nume, LENGTH(nume)-LENGTH(TRANSLATE(nume, 'DA', 'D')) "'A'"
```

```
FROM profesor;
```

NUME	'A'
-----	-----
GHEORGHIU	0
MARIN	1
GEORGESCU	0
IONESCU	0
ALBU	1
VOINEA	1
STANESCU	1



Funcții caracter - cont.

- **Observație:** În exemplul anterior, funcția TRANSLATE (nume, 'DA','D') va căuta în coloana nume primul caracter (caracterul 'D') din cel de-al doilea argument al funcției (șirul de caractere 'DA') și îl va înlocui cu primul caracter (adică tot cu caracterul 'D') din cel de-al treilea argument al funcției (șirul de caractere 'D'), apoi va căuta cel de-al doilea caracter, adică caracterul 'A', și îl va șterge din câmpul nume deoarece acesta nu are caracter corespondent în cel de-al treilea argument al funcției.
Am folosit acest artificiu deoarece șirul de caractere vid este echivalent cu valoarea Null deci funcția TRANSLATE (nume,'A','') ar fi înlocuit toate valorile câmpului nume cu valoarea Null.



Funcții numerice sau aritmetice

- Aceste funcții au ca argumente date numerice și returnează tot valori numerice.
- Marea majoritate a acestor funcții au o precizie de 38 de zecimale (COS, EXP, LN, LOG, SIN, SQRT, TAN au însă o precizie de 36 de zecimale).



Funcții numerice sau aritmetice - cont.

- Dintre cele mai importante funcții amintim:
 - ROUND - rotunjește valorile la un anumit număr de poziții zecimale;
 - TRUNC - trunchiază valorile un anumit număr de poziții zecimale;
 - CEIL - returnează cel mai mic întreg mai mare sau egal cu o anumită valoare;
 - FLOOR - returnează cel mai mare întreg mai mic sau egal cu o anumită valoare;
 - SIGN - returnează valoarea -1 dacă valoarea argumentului primit este mai mică decât 0, 1 dacă valoarea argumentului primit este mai mare decât 0 și 0 dacă valoarea argumentului primit este egală cu 0;



Funcții numerice sau aritmetice - cont.

- SQRT - returnează rădăcina pătrată a argumentului primit;
- ABS - returnează valoarea absolută a argumentului primit;
- POWER - returnează valoarea unui număr ridicat la o anumită putere;
- MOD - returnează restul împărțirii a două numere;
- alte funcții matematice cum ar fi: LOG, SIN, TAN, COS, EXP, LN.




Funcții pentru dată calendaristică și oră

- În Oracle datele de tip dată calendaristică sunt reprezentate sub un format numeric reprezentând: ziua, luna, anul, ora, minutul, secunda și secolul.
- Oracle poate manevra date calendaristice de la 1 ianuarie 4712 BC până la 31 decembrie 4712 AD.
- Modul predefinit de afișare și introducere este sub forma: DD-MON-YY (ex. 31-Dec-99). Aceasta categorie de funcții operează pe valori de tip dată calendaristică, rezultatul returnat fiind tot de tip dată calendaristică, excepție făcând funcția MONTHS_BETWEEN care returnează o valoare numerică.



Funcții pt. dată calendaristică și oră - cont.

- Cele mai des întâlnite funcții sunt:
 - `ADD_MONTHS` - returnează o dată calendaristică formată prin adăugarea la data calendaristică specificată a unui anumit număr de luni;
 - `LAST_DAY` - întoarce ca rezultat ultima zi a unei luni specificate;
 - `MONTHS_BETWEEN` - returnează numărul de luni dintre două date calendaristice specificate;
 - `NEXT_DAY` - returnează o dată ulterioară datei calendaristice specificate;
 - `SYSDATE` - întoarce ca rezultat data calendaristică a sistemului.
- Asupra datelor calendaristice se pot realiza operații aritmetice, cum ar fi scăderea sau adunarea, modul lor de funcționare fiind ilustrat în tabelul următor:

Tip operand	Operație	Tip operand	Tip Rezultat	Descriere
data 	+/-	Număr	data	Adaugă/scade un număr de zile la o dată calendaristică
data	+/-	număr/24	data	Adaugă/scade un număr de ore la o dată calendaristică
data	+/-	număr/1440	data	Adaugă/scade un număr de minute la o dată calendaristică
data	+/-	număr/86400	data	Adaugă/scade un număr de minute la o dată calendaristică
data	-	data	număr zile	Scade două date calendaristice rezultând diferența în număr de zile. Dacă al doilea operand este mai mare decât primul numărul de zile rezultat este reprezentat de o valoare negativă.



Funcții pt. dată calendaristică și oră - cont.

- De asemenea mai există funcțiile ROUND și TRUNC care rotunjesc, respectiv trunchiază data calendaristică.

Aceste funcții sunt foarte folositoare atunci când se dorește compararea datelor calendaristice care au ora diferită.

- Exemplul următor rotunjește data de naștere a cadrelor didactice în funcție de an:

<code>SELECT ROUND(data_nast, 'YEAR') "DATA"</code>	DATA
<code>FROM profesor;</code>	-----
	01-JAN-47
	01-JAN-45
	01-JAN-52
	01-JAN-66
	01-JAN-70



Funcții de conversie

- În general expresiile nu pot conține valori aparținând unor tipuri de date diferite. De exemplu, nu se poate înmulți 3 cu 7 și apoi aduna "ION". Prin urmare se realizează anumite conversii care pot fi implicite sau explicite.
- Conversiile implicite se realizează în următoarele cazuri:
 - atribuirii de valori unei coloane (folosind comenzile INSERT sau UPDATE) sau atribuirilor de valori unor argumente ale unei funcții;
 - evaluări de expresii.



Funcții de conversie - cont.

- Pentru atribuiri, Oracle efectuează în mod implicit următoarele conversii de tip:
 - VARCHAR2 sau CHAR la NUMBER
 - VARCHAR2 sau CHAR la DATE
 - VARCHAR2 sau CHAR la ROWID
 - NUMBER, DATE sau ROWID la VARCHAR2
- Conversia la atribuire reușește în cazul în care Oracle poate converti tipul valorii atribuite la tipul destinației atribuirii.
- Pentru evaluarea expresiilor, se realizează în mod implicit următoarele conversii de tip:
 - VARCHAR2 sau CHAR la NUMBER
 - VARCHAR2 sau CHAR la DATE
 - VARCHAR2 sau CHAR la ROWID



Funcții de conversie - cont.

- De exemplu, pentru următoarea interogare se realizează conversia în mod implicit a constantei de tip CHAR '10' la tipul NUMBER.

```
SELECT salariu + '10'
```

```
FROM profesor;
```

SALARIU+'10'

3010

2510

2810

1510

2210

1210

1210



Funcții de conversie - cont.

- Pentru conversiile explicite de tip, SQL pune la dispoziție mai multe funcții de conversie, de la un anumit tip de dată la altul, după cum este arătat în tabelul de mai jos.

	CHAR	NUMBER	DATE	RAW	ROWID
CHAR	-	TO_NUMBER	TO_DATE	HEXTORAW	CHARTOROWID
NUMBER	TO_CHAR	-	TO_DATE(nr, 'J')		
DATE	TO_CHAR	TO_DATE(dată, 'J')	-		
RAW	RAWTOHEX			-	
RAWID	RAWIDTOCHAR				-



Funcții de conversie - cont.

- Cele mai uzuale funcții sunt:
 - TO_CHAR - convertește un număr sau o dată calendaristică într-un șir de caractere;
 - TO_NUMBER - convertește un șir de caractere alcătuit din cifre la o valoare numerică;
 - TO_DATE - convertește un șir de caractere sau un număr ce reprezintă o dată calendaristică la o valoare de tip dată calendaristică.
De asemenea poate converti o dată calendaristică la un număr ce reprezintă data calendaristică Iuliană.
- Pentru a realiza conversia aceste funcții folosesc anumite măști de format.



Funcții de conversie - cont.

- Următorul exemplu va prelua data și ora curentă a sistemului din funcția SYSDATE și o va formata într-o dată scrisă pe litere ce va conține și ora în minute și secunde:

```
SELECT TO_CHAR(SYSDATE, 'DD MONTH YYYY HH24:MI:SS') Data  
FROM dual;
```

DATA

17 MAY 2000 17:03:38



Funcții diverse

- Acestea sunt în general funcții care acceptă ca argumente orice tip de dată.
- Cele mai utilizate sunt:
 - **DECODE** - Aceasta este una dintre cele mai puternice funcții SQL.
Ea practic facilitează interogările condiționate, acționând ca o comandă 'if-then-else' sau 'case' dintr-un limbaj procedural.
Pt. fiecare înregistrare se va evalua valoarea din coloana testată și se va compara pe rând cu fiecare valoare declarată în cadrul funcției.
Dacă se găsesc valori egale, atunci funcția va returna o valoare aferentă acestei egalități, declarată tot în cadrul funcției.
Se poate specifica ca, în cazul în care nu se găsesc valori egale, funcția să întoarcă o anumită valoare.
Dacă acest lucru nu se specifică funcția va întoarce valoarea Null.



Funcții diverse - cont.

- GREATEST - returnează cea mai mare valoare dintr-o listă de valori;
- LEAST - returnează cea mai mică valoare dintr-o listă de valori;
- VSIZE - returnează numărul de bytes pe care este reprezentată intern o anumită coloană;
- USER - returnează numele utilizatorului curent al bazei de date;
- DUMP - returnează o valoare ce conține codul tipului de dată, lungimea în bytes, precum și reprezentarea internă a unei expresii.

Funcții diverse - cont.

- Exemplul următor utilizează funcția DECODE pentru a returna o creștere a salariului cadrelor didactice cu grad de profesor, conferențiar și lector, restul salariilor rămânând nemodificate:

SELECT nume, grad, salariu,	NUME	GRAD	SALARIU	Salariu modificat
DECODE (grad, 'PROF', salariu*1.2,	-----	----	-----	-----
'CONF' , salariu*1.15,	GHEORGHIU	PROF	3000	3600
'LECT', salariu*1.1,	MARIN	PROF	2500	3000
salariu) "Salariu	GEORGESCU	CONF	2800	3220
modificat"	IONESCU	ASIST	1500	1500
FROM profesor;	ALBU	LECT	2200	2420
	VOINEA	ASIST	1200	1200
	STANESCU	ASIST	1200	1200



Funcții referitoare la mai multe înregistrări

- Aceste funcții se mai numesc și funcții totalizatoare sau funcții de grup. Spre deosebire de funcțiile referitoare la o singură înregistrare, funcțiile de grup operează pe un set de mai multe înregistrări și returnează un singur rezultat pentru fiecare grup. Dacă nu este utilizată clauza GROUP BY, ce grupează înregistrările după un anumit criteriu, tabela este considerată ca un singur grup și se va returna un singur rezultat.



Funcții referitoare la mai multe înreg. - cont.

- COUNT - determină numărul de înregistrări care îndeplinesc o anumită condiție;
- MAX - determină cea mai mare valoare dintr-o coloană;
- MIN - determină cea mai mică valoare dintr-o coloană;
- SUM - returnează suma tuturor valorilor dintr-o coloană;
- AVG - calculează valoarea medie a unei coloane;
- STDDEV - determină abaterea sau deviația standard a unei coloane numerice;
- VARIANCE - returnează dispersia, adică pătratul unei deviații standard pentru o coloană numerică.

Funcții referitoare la mai multe înreg. - cont.

➤ Exemplu:

```
SELECT MIN(salariu), MAX(salariu), AVG(salariu), COUNT(*)  
FROM profesor;
```

Toate funcțiile de mai sus, cu excepția funcției COUNT, operează asupra unei coloane sau unei expresii, care este specificată ca parametru al funcției.

În cazul funcției COUNT, argumentul acesteia nu contează, de obicei utilizându-se ca argument simbolul '*'.

MIN(SALARIU)	MAX(SALARIU)	AVG(SALARIU)	COUNT(*)
1200	3000	2057.1429	7



Funcții referitoare la mai multe înreg. - cont.

- **Observație:** Toate funcțiile de grup ignoră valorile Null, excepție făcând funcția COUNT. Pentru a include în calcule și înregistrările cu valoarea Null se poate folosi funcția NVL.
- Dacă nu este utilizată clauza GROUP BY, în lista de la SELECT nu pot apărea funcții de grup alături de nume de coloane sau alte expresii care iau valori pentru fiecare înregistrare în parte.
- De exemplu, următoarea interogare va genera o eroare:

```
SELECT nume, MIN(salariu)
FROM profesor;
ERROR at line 1:
ORA-00937: not a single-group group function
```



Pseudo-coloana ROWNUM

- ROWNUM este o pseudo-coloană care numerotează rândurile selectate de o interogare. Astfel, pentru primul rând selectat pseudo-coloana ROWNUM are valoarea 1, pentru al doilea rând are valoarea 2, ș.a.m.d.
- De exemplu, pentru a limita rândurile selectate de o interogare la maxim 5, se folosește următoarea comandă:

```
SELECT *  
FROM profesor  
WHERE ROWNUM < 6;
```




Pseudo-coloana ROWNUM - cont.

- Deoarece pseudo-coloana ROWNUM numerotează rândurile selectate, valorile sale vor trebui să înceapă tot timpul cu 1.
Deci dacă în exemplul anterior condiția ar fi $ROWNUM > 6$ sau $ROWNUM = 6$ interogarea nu ar selecta nici un rând deoarece în acest caz condiția ar fi întotdeauna falsă.
Pentru primul rând accesat de interogare ROWNUM va avea valoarea 1, condiția nu este îndeplinită și deci rândul nu va fi selectat.
Pentru al doilea rând accesat de interogare ROWNUM va avea din nou valoarea 1, condiția nu este îndeplinită nici în acest caz și deci nici acest rând nu va fi selectat.
Prin urmare nici unul din rândurile accesate nu vor satisface condiția conținută de interogare.



Pseudo-coloana ROWNUM - cont.

- Pseudo-coloana ROWNUM se poate folosi atât în condiția unor comenzi UPDATE sau DELETE cât și pentru a atribui valori unice unei coloane, ca în exemplul de mai jos:

```
UPDATE vanzari
```

```
SET cod = ROWNUM;
```

Valoarea pseudo-coloanei ROWNUM este atribuită unui rând înainte ca acesta să fie sortat datorită unei clauze ORDER BY, de aceea valorile pseudo-coloanei nu reprezintă ordinea de sortare.



Pseudo-coloana ROWNUM - cont.

➤ De exemplu

```
SELECT nume, prenume, ROWNUM  
FROM profesor  
ORDER BY salariu;
```

NUME	PRENUME	ROWNUM
-----	-----	-----
VOINEA	MIRCEA	6
STANESCU	MARIA	7
IONESCU	VERONICA	4
ALBU	GHEORGHE	5
MARIN	VLAD	2
GEORGESCU	CRISTIANA	3
GHEORGHIU	STEFAN	1



Clauza GROUP BY

- Clauza GROUP BY este utilizată pentru a împărți d.p.d.v. logic un tabel în grupuri de înregistrări.
Fiecare grup este format din toate înregistrările care au aceeași valoare în câmpul sau grupul de câmpuri specificate în clauza GROUP BY.
Unele înregistrări pot fi excluse folosind clauza WHERE înainte ca tabelul să fie împărțit în grupuri.
- Clauza GROUP BY se folosește de obicei împreună cu funcțiile de grup, acestea returnând valoarea calculată pentru fiecare grup în parte.
În cazul folosirii clauzei GROUP BY, toate expresiile care apar în lista de la SELECT trebuie să aibă o valoare unică pentru fiecare grup, de aceea orice coloană sau expresie din această listă care nu este o funcție de grup trebuie să apară în clauza GROUP BY.

Clauza GROUP BY - cont.

- Sunt prezentate în continuare câteva exemple:

SELECT grad, AVG(salariu)	GRAD	AVG(SALARIU)
FROM profesor	----	-----
GROUP BY grad;	ASIST	1300
	CONF	2800
	LECT	2200
	PROF	2750

SELECT grad, MAX(salariu)	GRAD	MAX(SALARIU)
FROM profesor	----	-----
WHERE prima is not NULL	ASIST	1200
GROUP BY grad;	CONF	2800
	LECT	2200
	PROF	3000



Clauza GROUP BY - cont.

- Următoarea interogare va genera o eroare deoarece în lista de SELECT există o coloană (nume) care nu apare în clauza GROUP BY:

```
SELECT nume, MIN(salariu)
```

```
FROM profesor
```

```
GROUP BY grad;
```

```
ERROR at line 1:
```

```
ORA-00979: not a GROUP BY expression
```

- Comanda de mai sus este invalidă deoarece coloana nume are valori individuale pentru fiecare înregistrare, în timp ce MIN(salariu) are o singură valoare pentru un grup.



Clauza GROUP BY - cont.

- Clauza GROUP BY permite apelarea unei funcții de grup în altă funcție de grup.
- În exemplul următor, funcția AVG returnează salariul mediu pentru fiecare grad didactic, iar funcția MAX returnează maximul dintre aceste salarii medii.

```
SELECT MAX (AVG (salariu) )
```

```
FROM profesor
```

```
GROUP BY grad;
```

```
MAX(AVG(SALARIU))
```

```
-----
```

```
2800
```



Clauza HAVING

- Clauza HAVING este tot o clauză de filtrare ca și clauza WHERE. Totuși, în timp ce clauza WHERE determină ce înregistrări vor fi selecționate dintr-un tabel, clauza HAVING determină care dintre grupurile rezultate vor fi afișate după ce înregistrările din tabel au fost grupate cu clauza GROUP BY. Cu alte cuvinte, pentru a exclude grupuri de înregistrări se folosește clauza HAVING, iar pentru a exclude înregistrări individuale se folosește clauza WHERE.
- Clauza HAVING este folosită numai dacă este folosită și clauza GROUP BY. Expresiile folosite într-o clauză HAVING trebuie să aibă o singură valoare pe grup.



Clauza HAVING - cont.

- Atunci când se folosește clauza GROUP BY, clauza WHERE se utilizează pentru eliminarea înregistrărilor ce nu se doresc a fi grupate.
- Astfel, următoarea interogare este invalidă deoarece clauza WHERE încearcă să excludă grupuri de înregistrări și nu anumite înregistrări:

```
SELECT grad, AVG(salariu)
FROM profesor
WHERE AVG(salariu) > 2000
GROUP BY grad;
```

ERROR:

ORA-00934: group function is not allowed here



Clauza HAVING - cont.

- Pentru a exclude gradul didactic pentru care media de salariu nu este mai mare decât 2000 se folosește următoarea comandă SELECT cu clauza HAVING:

```
SELECT grad, AVG(salariu)
FROM profesor
GROUP BY grad
HAVING AVG(salariu) > 2000;
```

GRAD	AVG(SALARIU)
------	--------------

CONF	2800
LECT	2200
PROF	2750



Clauza HAVING - cont.

- Următoarea interogare exclude întâi cadrele didactice care nu au salariu mai mare decât 2000 după care exclude gradul didactic pentru care media de salariu nu este mai mare decât 2500.

```
SELECT grad, AVG(salariu)
FROM profesor
WHERE salariu > 2000
GROUP BY grad
HAVING AVG(salariu) > 2500;
```

GRAD	AVG(SALARIU)
----	-----
CONF	2800
PROF	2750



Bibliografie

F. Ipate, M. Popescu, Dezvoltarea aplicațiilor de baze de date în Oracle 8 și Oracle Forms 6, Editura ALL, 2000.