2. Baze de date, sisteme de baze de date, etapele realizarii unei baze de date

Baza de date

= un ansamblu structurat de date coerente, fără redundanţă inutilă, astfel încât acestea pot fi prelucrate eficient de mai mulţi utilizatori într-un mod concurent

= o colecţie de date persistente, care sunt folosite de către sistemele de aplicaţii ale unei anumite „întreprinderi“

Sistem de baze de date = consta din:
1.BD propriu-zisă (în care se memorează datele)
2.SGBD (gestionarea şi prelucrarea complexă a datelor)
3. dicţionarul BD (metabaza de date: informaţii despre date, structura acestora, statistici, documentaţie)
4.mijloace *hardware* (comune sau specializate);
5.reglementări administrative destinate bunei funcţionări a sistemului
6.personalul implicat:
•administratori de date şi baze de date,
•proiectanţi (designeri) de baze de date,
•programatori de aplicaţii,
•utilizatori finali.

Administratorul de date:
= un manager care stabileste
1)care sunt datele ce trebuie stocate
2)regulile de intretinere si de tratare a acestor date
Administratorul bazei de date:
= o persoană sau un grup de persoane ce răspund de ansamblul activităţilor legate de BD

- creează baza de date reală,
- implementează elementele tehnice de control,
- asigura funcţionarea sistemului la performanţe adecvate, monitorizeaza performanţele BD,
- furnizează diverse servicii tehnice

este responsabil cu implementarea deciziilor DA şi cu controlul general al sistemului, la nivel tehnic => are 4 mari categorii de atribuţii:
•de proiectare,
•administrative,
•operative,
•de coordonare.

Proiectanţii de BD: (i) cei care abordează nivelul logic:
•proiectează conceptual baza de date (independent de programele de aplicatii si limbajele de programare) (ii) cei care abordează nivelul fizic:
•aleg modul de implementare fizica a modelului conceptual

Programatorii de aplicaţii:
•scriu programele aplicaţie ce conferă funcţionalitatea cerută de utilizatorii finali
•utilizeaza limbaje de programare de nivel inalt (*C++*, *Java, PL/SQL* etc.).

Utilizatorii finali:
•accesează interactiv baza de date
•pot fi:
utilizatori simpli
utilizatori sofisticaţi

Etapele realizarii unei BD

1) Analiza situatiei existente
2) Priectarea BD
3) Implementarea

1) a) examinam sistematic si aprofundat acel aspect din viata reala
b)I)gradul de generalitate(scope-ul)
II)dimensiunea BD
III)cateogriile si numarul de viitori useri.

2) 3 etape:
I)proiectare la nivel conceptual
II) proiectare la nivel logic
III) proiectare la nivel fizic
Aici imi selectez SGBD-ul, se fac interfetele si aplicatiile.

3)Realizarea propriu-zisa(scrierea programelor)

- Conversia + incarcarea datelor;
- Testearea prototipului
- Implementarea propriu-zisa

3. Clasificarea BD

Criterii de clasificare a BD:
1)modelul de date

- Prerelational
- Relational
- Postrelational

2)nr de useri

- Cele mai frecvente
- Permit accesul concurrent la BD;
- Pot fi si sisteme monoutilizator

3)nr. De calc pe care sunt sticate BD si SGBD
•centralizate:
•datele si SGBD sunt stocate pe o singura statie (calculator)
•distribuite:
•şi datele si SGBD sunt distribuite pe mai multe calculatoare interconectate printr-o retea de comunicatie.

4)modul de functionare
•teleprocesarea
•arhitectura tradiţională: 1! calculator cu 1! unitate CPU şi
•un numar de terminale, incapabile să funcţioneze singure;

•arhitectura fişier-server

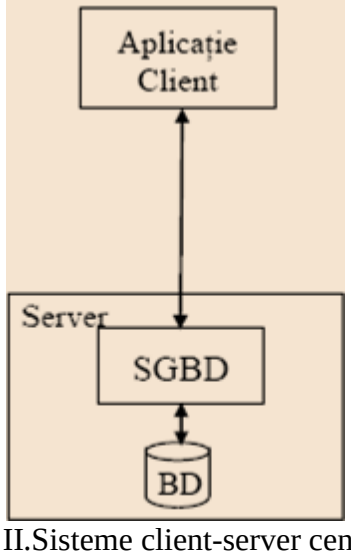
•procesarea este distribuită în reţea (de obicei LAN)
•arhitectura cuprinde fişierele cerute de aplicaţii şi SGBD-ul
•aplicaţiile şi funcţiile SGBD sunt executate pe fiecare staţie de lucru, solicitând atunci când este nevoie fişiere de pe serverul de fişiere;

•arhitectura client-server există

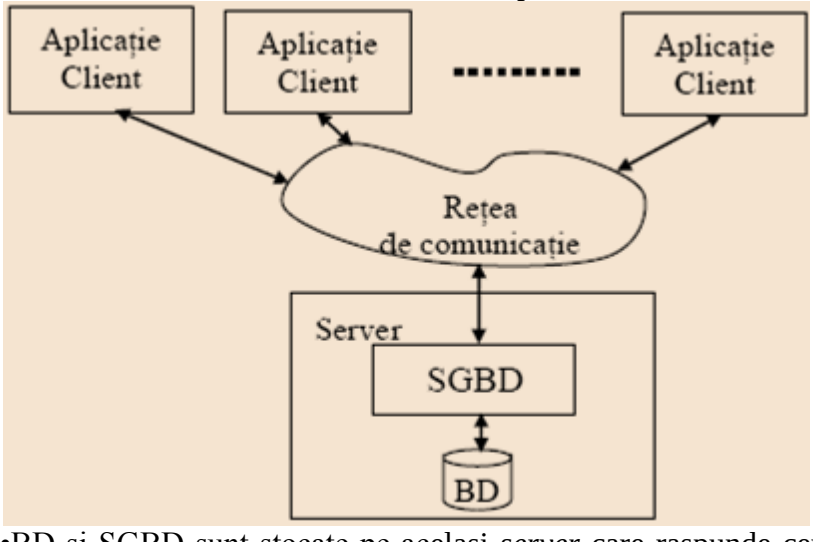
•un proces client, care necesită resurse şi
•un proces server, care oferă resurse.

5) implementarea sistemului de BD.
se combina ultimele 3 criterii : numarul de utilizatori, modul de stocare a BD şi SGBD şi modul de functionare a sistemului de baze de date:
I.sisteme client-server centralizate de tip *monouser*
II.sisteme client-server centralizate de tip *multiuser*
III.sisteme client-server distribuite de tip *multiuser*.

I.Sisteme client-server centralizate de tip *monouser*
•BD şi SGBD sunt stocate pe acelasi *server* care raspunde cererilor unui singur *client* care acceseaza BD



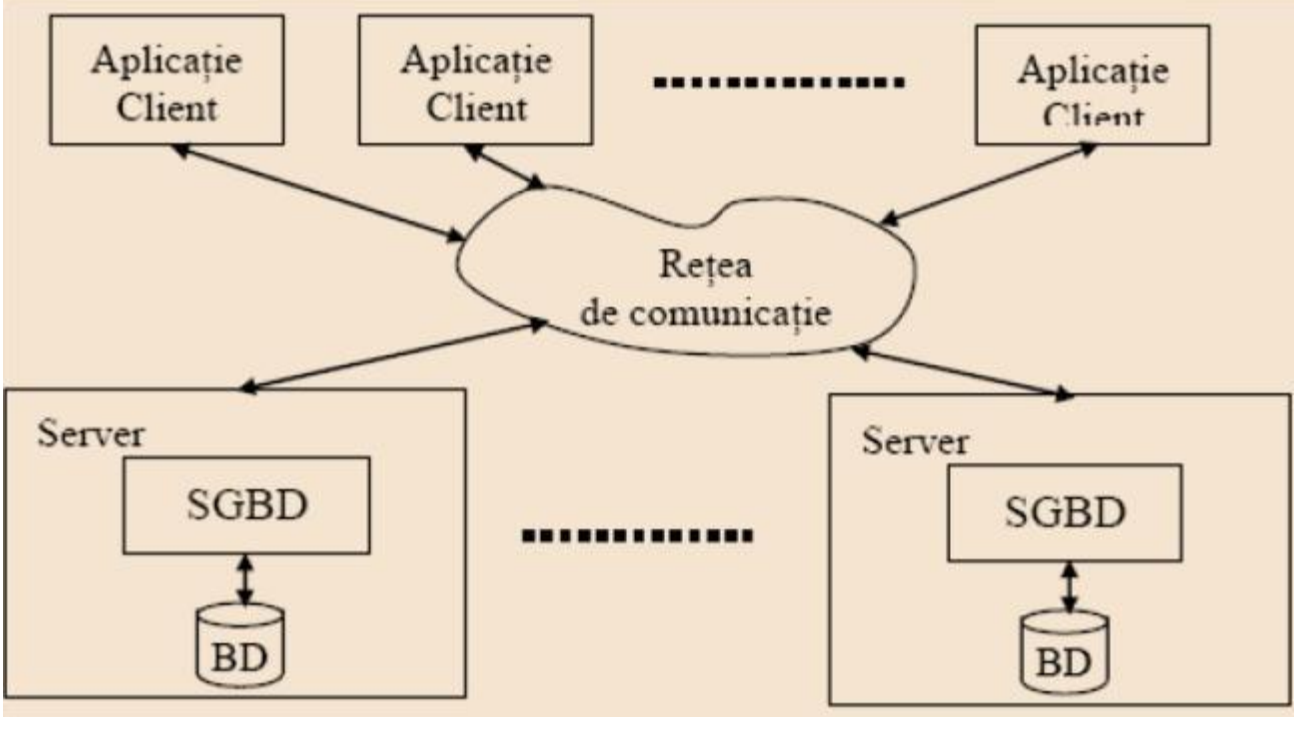
II.Sisteme client-server centralizate de tip *multiuser*



•BD şi SGBD sunt stocate pe acelasi *server* care raspunde cererilor mai multor *clienti* care acceseaza BD

•aplicatiile *client* sunt executate pe statii diferite (=> cu puteri de calcul inferioare *serverului*), conectate printr-o retea de comunicatie cu calculatorul pe care ruleaza *serverul*.

III.Sisteme client-server distribuite de tip *multiuser*
•**O BD distribuita** = = o colectie de date care, din punct de vedere logic, apartin aceluiasi sistem dar care, din punct de vedere fizic, pot sa fie memorate pe mai multe statii de calcul conectate printr-o retea de comunicatie
•**SGBD distribuit** = = sistemul software care gestioneaza o astfel de BD
•Caracteristici:
•cresterea capacitatii destocare şi prelucrare
•cresterea complexitatii
•Principala cerinta (partial indeplinita): **transparenta** = capacitatea unui sistem distribuit de a ascunde detaliile de implementare, astfel încât utilizatorii sa poata accesa datele pe baza unui model de nivel înalt, fara a fi necesara cunoasterea exacta a modului de amplasare, replicare sau comunicare a datelor.



- BD şi SGBD sunt distribuite pe mai multe statii conectate printr-o retea de comunicatie

•aplicatiile *client* sunt executate pe statii diferite , conectate printr-o retea de comunicatie cu calculatoarele (interconectate) pe care ruleaza *serverul*.

4. Sisteme de gestiune a bazelor de date (definitie, arhitectura, obiective)

Sistem de gestiune a bazelor de date (SGBD – *Data Base Management System*)
= un produs *software* care asigură interacţiunea cu o BD, permiţând definirea, consultarea şi actualizarea datelor din BD

Structura unui SGBD:
•complexa; dinamica; minimum 5 clase de module:
1.programe de gestiune a bazei de date (PGBD): realizează accesul fizic la date ca urmare a unei comenzi;

2.module pentru tratarea LDD permit traducerea unor informaţii despre date în obiecte ce pot fi apoi exploatate în manieră procedurală / neprocedurală;
3.module pentru tratarea LMD (interpretativ, compilativ, generare de programe) permit utilizatorilor inserarea, ştergerea, reactualizarea sau consultarea informaţiei dintr-o bază de date;
4.module utilitare asigură întreţinerea, prelucrarea, exploatarea corectă şi uşoară a bazei de date;
5.module de control
•permit controlul programelor de aplicaţie,
•asigurarea confidenţialităţii şi integrităţii datelor,
•rezolvarea unor probleme de concurenţă, r
•ecuperarea informaţiei în cazul unor avarii sau defecţiuni *hardware* sau *software* etc.

Cele **4 niveluri de abstractizare şi de percepţie** a datelor intr-o BD:

LOW

- Intern → nivel la care datele exista efectiv

- Conceptual → aceste nivele reprezinta numai
- Logic → virtualizari ale datelor care exista
- Extern → doa r la nivel intern

HIGH
=> arhitectura pe 3 niveluri a BD si existenta unor corespondente intre acestea.

Nivelul extern (modelul extern, subschema, vizualizarea)
•reprezintă viziunea utilizatorului final asupra datelor
•permite asigurarea unui nivel de securitate a datelor: un utilizator va accesa doar datele descrise în schema sa externă Nivelul logic (una din schemele logice posibile ale datelor)
•reprezintă viziunea programatorului de aplicaţie asupra datelor; Nivelul conceptual (schema conceptuală a datelor: articol, înregistrare, zonă)
•este nivelul central
•reprezintă viziunea programatorilor de sistem asupra datelor
•corespunde structurii semantice a datelor fără implementarea pe calculator
Nivelul intern (schema fizică a datelor: bit, octet, adresă)
•permite descrierea datelor unei BD sub forma în care sunt stocate în memoria calculatorului
•sunt definite fişierele care conţin aceste date, articolele din fişiere, căile de acces la aceste articole etc.

Observatie
•La nivel conceptual sau intern:
•schemele respective descriu in mod unic o bază de date
La nivel extern:
•schemele reprezintă o descriere a unei părţi a bazei de date ce corespunde viziunii unui program sau unui utilizator =>Pentru o BD particulară există: 1! schemă internă, 1! schemă conceptuală mai multe scheme externe.

OBIECTIVE:

SGBD (gestionarea şi prelucrarea complexă a datelor)
5. Structura fizica a unei BD Oracle (fişiere de date,fişiere de reluare, fişiere de control)

Structura fizică a bazei de date *Oracle:*
A.fişiere de date (*Datafiles*),
B.fişiere de reluare (*Redo Log Files*),
C.fişiere de control (*Control Files*);

A. Fişierele de date = fişiere fizice ale SO
•stochează datele tuturor structurilor logice ale bazei;
•alocarea unui fişier de date bazei *Oracle: SO*
•şterge informaţiile nefolosite
•acordă autorizaţii pentru fisier;
•primul fişier de date creat: fisierul care stochează dicţionarul datelor.

B. Fişierele de reluare = înregistrează toate modificările care
•au loc asupra datelor bazei (indiferent dacă au fost permanentizate sau nu) şi
•nu au fost scrise încă în fişierele de date;
•sunt specificate în momentul creării sau modificării bazei
•sunt utilizate în manieră circulară (cele care au fost folosite în întregime, pot fi arhivate până când sistemul le va reutiliza)
•asigură protecţia BD în cazul defecţiunilor
•o BD *Oracle* conţine două sau mai multe fişiere de reluare.

C. Fişierele de control = fişiere binare de dimensiune redusă, necesare pentru pornirea şi funcţionarea bazei de date;
•orice BD *Oracle* deţine cel puţin un fişier de control;
•fiecare fişier de control
•este asociat unei singure BD
•conţine informaţii despre structura fizică a acesteia
•este creat odată cu respectiva BD (*Oracle* permite existenţa fişierelor de control multiplexate

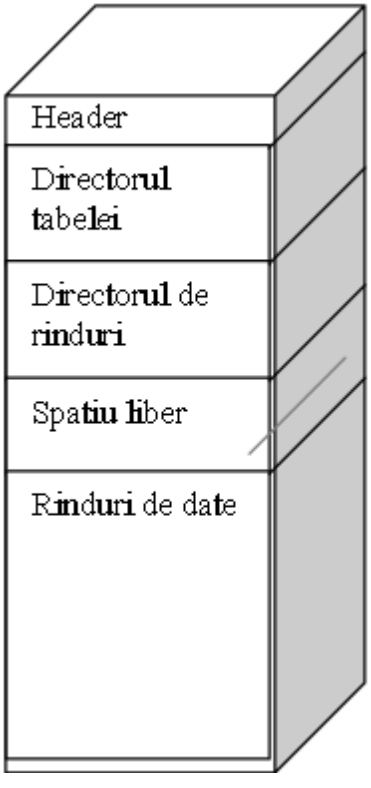
•La pornirea unei instanţe *Oracle:*
•sistemul foloseşte fişierul de control pentru:
•a identifica baza şi
•a determina dacă aceasta este în stare validă pentru utilizare;
•sunt identificate fişierele de reluare necesare execuţiei operaţiilor bazei de date;
•Fişierele de control reflectă automat schimbările (creare, redenumire sau ştergere) care au loc la nivelul fişierelor de date sau de reluare;
•Informaţiile din fişierele de control pot fi modificate doar de serverul *Oracle*;

6. Structura logica a unei BD Oracle (blocurile de date, extensiile, segmentele, spaţiile tabel, obiectele schemei)

Structura logică a bazei de date *Oracle:*
a.blocurile de date (*data block*),
b.extensiile (*extent*),
c.segmentele (*segment*),
d.spaţiile tabel (*tablespace*),
e.obiectele schemei (*schema object*).
a)Blocuri de date = unitati logice prin care sistemul administrează spaţiul de stocare al fişierelor de date; Blocul = cea mai mică unitate I/O folosită de baza de date, = corespunzătoare unui bloc fizic de

octeţi de pe disc, = dimensiunea sa: este definită în momentul creării BD, poate fi modificată ulterior, este un multiplu al dimensiunii blocurilor fizice de la nivelul SO; Structura blocului de date *Oracle*:
•un antet (*header*),
•un spaţiu liber (*free space*),
•un spaţiu pentru date (*data space*).

Blocuri de date (cont.)
•**Antetul** conţine
•informaţii generale referitoare la bloc
•un catalog al tabelelor (*table directory*):
•un catalog al liniilor (*row directory*):



Spaţiul liber al blocului de date este alocat pentru inserarea de noi linii sau actualizarea liniilor care necesită spaţiu suplimentar.
•Alegerea blocului în care va fi inserată o linie nouă depinde de spaţiul liber al acestuia şi de valorile parametrilor *PCTFREE* şi *PCTUSED*.
•Într-un bloc, se pot introduce date atâta timp cât dimensiunea spaţiului liber este mai mare decât limita fixată de parametrul *PCTFREE*. Sistemul *Oracle* va considera acest bloc indisponibil pentru inserarea de noi linii, până când procentajul spaţiului utilizat coboară sub valoarea dată de *PCTUSED*.

b) c) Extensia şi segmentul
□□**Extensia** = unitate logica de alocare a spaţiului BD,
= compusă dintr-o mulţime contiguă de blocuri de date (din acelaşi fişier de date);
□□**Segmentul** = unitate logica formata din una sau mai multe extensii;
Iniţial, segmentul are o singură extensie (*initial extent*).

b) c) Extensia şi segmentul (cont.) **O extensie**
•este alocată atunci când este creat sau extins un segment,
•este dezalocată (in general) când segmentul este suprimat sau trunchiat;
•Eliberarea unei extensii implică ştergerea datelor existente în blocurile de date alocate acesteia (ele vor fi reutilizate pentru extensiile nou create);

b) c) Extensia şi segmentul (cont.) □□**Segmentul** = corespunde unui singur obiect fizic stocat = foloseşte blocuri de date care se găsesc în acelaşi spaţiu tabel; Tipuri de segmente din BD *Oracle* :
•segmente de date (*data segment*),
•segmente index (*index segment*),
•segmente temporare (*temporary segment*),
•segmente de revenire (*undo segment*) etc.

b) c) Extensia şi segmentul (cont.) **Segmentele de date**
•sunt definite atunci când este folosită comanda de creare a unui tabel sau a unei grupări
•un singur segment de date este folosit pentru stocarea tuturor datelor dintr-un tabel nepartiţionat care nu face parte din nicio grupare, dintr-o partiţie a unui tabel partiţionat sau dintr-o grupare de tabele
Segmentele index
•sunt folosite pentru a stoca datele unui index

•fiecare index nepartiţionat este conţinut într-un singur segment. În cazul indecşilor partiţionaţi, fiecărei partiţii i se asociază câte un segment index **Segmentele temporare**
•sunt utilizate de sistem pentru analiza şi execuţia comenzilor *SQL* care necesită un spaţiu temporar de stocare
•sistemul alocă în mod automat segmente temporare atunci când este necesar şi le suprimă după execuţia comenzii *SQL*
•segmentele temporare sunt alocate în majoritatea cazurilor de sortare (atunci când operaţia respectivă nu se poate face în memorie sau dacă folosirea indecşilor nu presupune o soluţie mai eficientă).
b) c) Extensia şi segmentul (cont.) **Segmentele de revenire**
O BD conţine unul sau mai multe segmente de revenire, folosite pentru:
•anularea acţiunii tranzacţiilor
•asigurarea consistenţei la citire,
•efectuarea operaţiile de recuperare a bazei de date;
Segmentele de revenire **nu pot fi** accesate de către utilizatorii sau administratorii bazei de date
Ele pot fi scrise şi citite **doar** de către sistem.

d) Spatiul tabel = unitate logica de stocare formata din 1,2,… segmente
•grupeaza logic o mulţime de obiecte:
•fiecare obiect al BD are specificat un spaţiu tabel în care trebuie să fie creat ->
•datele care alcătuiesc obiectul sunt apoi stocate în fişierele de date alocate spaţiului tabel respectiv ->
•un fişier de date poate fi alocat unui singur spaţiu tabel;
•fiecarui utilizator i se poate aloca explicit un spaţiu tabel, în care vor fi stocate toate obiectele create de el
•alocarea se efectueaza automat
•folosirea mai multor spaţii tabel -> flexibilitate în utilizarea BD
•BD = {**spaţii tabel**} Tipuri de spatiu tabel in BD *Oracle:*
•spaţiul tabel *SYSTEM,*
•spaţii tabel non-*SYSTEM.*

e) Schema = mulţimea obiectelor bazei de date, aflate în posesia unui utilizator (fiecare utilizator deţine o singură schemă).
•numele schemei este acelaşi cu numele utilizatorului
•nu există o corespondenţă biunivocă intre spaţiile tabel şi schemele de obiecte
•obiectele aceleiaşi scheme pot fi în spaţii tabel diferite
•un spaţiu tabel poate conţine obiecte din mai multe scheme
 pentru a accesa un obiect din propria schemă, utilizatorul poate folosi doar numele acestuia
 pentru referirea unui obiect din schema altui utilizator, trebuie specificat atât numele obiectului, cât şi schema din care face parte, prin folosirea notaţiei *schema.obiect*

7. Structura unei BD Oracle: dictionarul datelor
Dictionarul datelor (catalogul de sistem)
= conţine „date despre date“ (metabaza de date) i.e. informaţii despre baza de date:
•definiţiile tuturor obiectelor din schemele bazei
•cantitatea de spaţiu alocat pentru obiectele schemelor
•cantitatea de spaţiu utilizat de acestea la momentul curent
•valorile implicite ale coloanelor
•constrângerile de integritate
•numele utilizatorilor *Oracle*
•privilegiile şi *role*-urile acordate fiecărui utilizator

•informaţii de auditare etc.

Dictionarul datelor (cont.)
•este generat automat la crearea BD;
•este reactualizat de către serverul *Oracle* după fiecare comandă LDD sau LCD;
•conţinutul său reflectă imaginea bazei de date (structura fizică şi logică) la un moment dat;
•din punct de vedere structural este compus:
•tabele de bază ale dictionarului şi
•vizualizări publice asupra acestora; => "vizibil" şi pt sistem şi pt utilizatori.

Dictionarul datelor (cont.)
 Tabelele de bază
•stochează informaţiile asociate BD,
•sunt primele obiecte create;
 Vizualizările
•decodifică informaţiile stocate în tabelele de bază şi
•le sintetizează pentru a fi disponibile utilizatorilor;

- este deţinut de către utilizatorul *SYS* şi se află în spaţiul tabel *SYSTEM;*

•sistemul poate accesa dicţionarul datelor pentru a obţine informaţii despre:
•utilizatori,
•obiecte,
•structurile de stocare;
•orice utilizator poate consulta dicţionarul datelor pentru a afla informaţii despre baza de date (documentare sau administrare)
•utilizatorii fără privilegii de administrare pot accesa doar vizualizările prefixate de *USER_* sau *ALL_*
•pentru a obţine lista vizualizărilor disponibile se poate interoga vizualizarea *DICTIONARY* care are sinonimul *DICT*
•se utilizeaza instrucţiunea *SELECT* din *SQL*.

8. Arhitectura interna a sistemului *Oracle* (arhitectura proceselor: procese user, procese Oracle (procese server, procese background))

Arhitectura proceselor Observatie Pentru a accesa o instanţă a unei BD *Oracle*, se execută:
•o aplicaţie sau un utilitar *Oracle* (prin intermediul cărora se lansează comenzi *SQL* asupra bazei – de ex. *Recovery Manager, Oracle Entreprise Manager, Oracle Forms*)
•un cod *Oracle* server (cu ajutorul căruia sunt interpretate şi procesate comenzile *SQL*);

•Un proces = un mecanism al sistemului de exploatare care permite executarea unor operaţii de calcul sau operaţii I/O;
•Fiecărui proces i se alocă o zonă privată de memorie
•Serverul *Oracle* : două tipuri generale de procese:
•procese *user*: execută aplicaţiile,
•procese *Oracle* (procese *server* şi *background*): asigură gestiunea informaţiilor dintr-o bază de date.

Un proces *user*
•creat de sistemul *Oracle* pentru:
•a executa codul unei aplicaţii program sau
•ca urmare a lansării unui utilitar *Oracle*

•se execută pe maşina client
•începe şi se termină odată cu aplicaţia utilizatorului resp.;
•nu interacţionează în mod direct cu serverul *Oracle* ci generează mesaje printr-un program interfaţă (*UPI = User Program Interface*).

Un proces *Oracle* = execută instrucţiunile interne ale serverului *Oracle*
•este invocat de alte procese pentru a îndeplini anumite operaţii în favoarea acestora;
•două tipuri de procese *Oracle:*
•procese server (*server process*),
•procese de fundal (*background process*).

Procesele *Oracle*
•Un proces *server*
•interacţionează cu procesele *user* ,
•comunică în mod direct cu serverul *Oracle* pentru a transmite cererile acestora printr-un program de interfata Oracle (*OPI = Oracle Program Interface*),
•este lansat când utilizatorul iniţiază o sesiune.

Procesele *Oracle*
•Un proces de fundal (*background process*)
• reuneste funcţiile executate pentru fiecare proces *user*
• execută operaţiile I/O asincrone,
• monitorizează alte procese *Oracle;*
•folosit pentru a îmbunătăţi performanţele unui sistem multiprocesor, în prezenţa mai multor utilizatori,
•serverul *Oracle* creează câte un set de procese *background* pentru fiecare instanţă

9. Arhitectura interna a sistemului *Oracle* (arhitectura memoriei (SGA, PGA))
Arhitectura interna a sistemului *Oracle* *(cont.)*
•**Arhitectura memoriei**
•structural, memoria este compusă din:
•o zona de memorie partajata = zona globală sistem (*SGA = System Global Area*),
•o zona de memorie nepartajata = zona globală program (*PGA = Program Global Area*);
•toate structurile de memorie se găsesc în memoria centrală,
•sunt create şi utilizate pentru a depozita:
•codul programelor executate,
•datele necesare în timpul execuţiei acestora,
•datele folosite în comun de mai multe procese *Oracle,*
•informaţiile referitoare la sesiunile curente etc**.**

SGA **=** *System Global Area = zona globala sistem* = **=** este un grup de structuri partajate de memorie care conţin date şi informaţii de control relative la BD şi la o instanţă;
•fiecare instanţă are propria sa *SGA* care:
•este alocata atunci când este pornita instanţa
•este eliberata în momentul opririi instantei;
•datele conţinute în *SGA* sunt folosite în comun de către utilizatorii conectaţi la instanţă,
•informaţiile conţinute în SGA sunt repartizate în diferite zone (*database buffer cache, redo log buffer*, *shared pool* etc.), care sunt alocate la pornirea instanţei;

•***SGA* fixă** = o zona specială a *SGA* folosită pentru stocarea informaţiilor despre starea bazei de date şi a instanţei
•informaţiile sunt accesate de către procesele *background,*
•nu poate conţine date ale utilizatorilor**.**

***PGA* = *Program Global Area = zona globala program* = =** zonă de memorie care conţine date şi informaţii de control relative la un singur proces *Oracle*
•poate fi folosită de un singur proces,
•este alocată la crearea procesului,
•este dezalocată la terminarea acestuia,
•este formată - in general – din:
•o zonă privată *SQL* (conţine date - de ex., informaţii de legătură şi structuri de memorie necesare rulării comenzilor)
•o zonă de memorie alocată sesiunii,
•zone de lucru *SQL*.

10. Modelarea semantică a informaţiei (modelul şi diagrama entitate-relatie E/R)

Modelul entitate-relatie (modelul E/R)
= una dintre cele mai cunoscute şi utilizate abordări ale modelării semantice (= una din primele etape în proiectarea BD, etapa numita proiectarea schemei conceptuale)
Metodologia E/R: considerata:
•cea mai buna metodologie pentru proiectarea BD
•una dintre cele mai bune metodologii pentru dezvoltarea

ModelulE/R
= model de date conceptual de nivel înalt, independent de platforma *hardware*utilizată şi de tipul SGBD-ului
•constituit din concepte care descriu
•structura BD şi
•tranzacţiile de regăsire sau reactualizare asociate
•împarte elementele unui sistem real în două categorii:
•entităţi
•relaţii (legături, asocieri, nu concept matematic) între aceste entităţi;
•entităţiile şi legăturile au anumite caracteristici, numite atribute.

Diagramele E/R
= reprezentare grafică a modelului E/R
= o tehnică de reprezentare grafica a structurii logice a BD;
Conventii de reprezentare in diagrama E/R1:
1.entităţile sunt reprezentate prin dreptunghiuri;
2.relaţiile dintre entităţi sunt reprezentate prin arce neorientate;
3.cardinalitatea minimă este indicată în paranteze, iar cardinalitatea maximă se scrie fără paranteze;
4.atributele care reprezintă chei primare trebuie subliniate sau marcate prin simbolul „#“, plasat la sfârşitul numelui acestor atribute; atributele obligatorii/optionale sunt precedate de */o;
5.nu este necesar să fie specificate, în cadrul diagramei, toate atributele.

Algoritm de proiectare a diagramei E/R
1.reprezentarea entităţilor din cadrul sistemului analizat;

2.reprezentarea relaţiilor (asocierilor) dintre entităţi şi a cardinalităţii;
3.reprezentarea atributelor aferente entităţilor şi relatiilor dintre entităţi;
4.evidentierea atributelor de identificare a entităţilor, adică a cheilor.

11. Modelul şi diagrama entitate-relatie extinse (specializare, generalizare, mostenire, restrictii în ierarhia Is-A: definitii, exemplificari)

Modelul Entitate-Relatie Extins (*Enhanced Entity-Relationship Model* = ***E-E/R model***) =
= permite definirea de ierarhii de clase de entitati prin specializare şi generalizare
Observatie

Cele 2 procese de abstractizare a datelor:
•au ca punct de plecare valorile unui/mai multor atribute clasificatoare in raport cu entitatile modelului E-E/R
•nu sunt neaparat inverse unul celuilalt;

Modul de reprezentare grafica:
•diagrama entitate-relatie extinsa
Specializare =
= proces de abstractizare a datelor prin care, pornind de la o entitate data, se definesc una sau mai multe subentitati, diferentiate între ele în functie de:
•rolul specific pe care îl au în modelul de datesau
•valorile unui/unor atribute clasificatoare;

Exemple
1.fie entitatea PERSONAL_FMI; din ea se pot defini prin specializare subentitatile: PERSONAL_DIDACTIC, PERSONAL_TEHNIC şi PERSONAL_ADMINISTRATIV in conformitate cu atributiile (**rolurile**) pe care angajatii FMI le au în cadrul facultatii,
2.fie entitatea UNITATE_ADMINISTRATIVA si **atributul *tip***=> se pot defini subentitatile: SAT, COMUNA, ORAS, MUNICIPIU, SECTOR, JUDET, fiecare având atributele sale proprii;
Observatie

Entitate= clasa; subentitate = subclasa.

Generalizare =
= procesul de abstractizare a datelor prin care se creeaza o supraentitate pornind de la mai multe entitati care au unul sau mai multe atribute comune
Exemplu
din entitatile TABLET_PC, NOTEBOOK , LAPTOP, DESKTOP, MAINFRAME se poate defini prin generalizare supraentitatea CALCULATOR ELECTRONIC
Observatii
Entitati= clase; supraentitate = supraclasa.

Ierarhia de clase *ISA* =
•în modelul E-E/R, subclasele, clasele, supraclasele formeaza o ierarhie de clase;
•între o subclasa (subentitate) şi o supraclasa (supraentitate) există o relaţie tipică:
•numită *ISA*
•de cardinalitatea maximă 1:1 şi
•de cardinalitate minimă 1:0

Observatie
•Reflexivitate şi tranzitivitate, nu şi simetrie!
•Clasele se aliniază în diagrama E-E/R pe verticala.

Modelul E-E/R este un model de date mult mai general care poate fi transpus în diferite modele de date specializate, inclusiv modelul OO.

Observatii
•pentru noile supraentităţi sunt necesare, uneori, chei primare artificiale1
•pe langa atributele care le clasifica, subentitatile au şi alte atribute specifice rolului lor in model2

Observatie
•instanţele unei supraclase includ toate instanţele subclaselor sale directe, precum şi toate instanţele subclaselor acestora =>
•exista instante înBD care pot fi vazute simultan la niveluri diferite în ierarhia *ISA*. =>
•conceptul de mostenire din ierarhia *ISA* permite ca atributele comune la nivelul unei clase şi la nivelurile subclaselor acesteiasă fie exprimate la nivelul cel mai comun de supraclasă, în loc sa fie repetate la fiecare nivel al ierarhiei de clase=>
•atributele sunt moştenite descendent în ierarhie NU ŞI ascendent
Exemplu:
un cadru didactic (i.e. PERSONAL_DIDACTIC, considerat subentitate a entităţii PERSONAL),are ca atribut titlul stiintific de doctor; acest atribut nu este semnificativ (desi in realitate poate exista) pentru un inginer (care face parte din PERSONAL_TEHNIC); cheia primară a subentităţii PERSONAL_DIDACTIC va fi*CNP*, care este si cheia primară a supraentităţii PERSONAL.

12. Modelul relational (regulile lui Codd, fundamentarea matematica, structura relationala a datelor, operatori)
DefiniţieRelaţie
= se numeşte **relaţie** peste mulţimileM1, M2, …Mn orice submulţime a produsului lor cartezian:
R M1, x M2, x ... x Mn.
Exemplu
Fie mulţimile
Marca= {Dacia, Ford, Fiat, Audi, Opel, Volvo},
Tip= {benzină, motorină}
CapacCil= {1100, 1200, 1300, 1400, 1600},
NrLoc= {4,5},
NrUşi= {2, 4, 5}.
Atunci, entitatea*Automobil*poatefi reprezentatăca o relaţiepeste acestemulţimi:
Automobil Marcax Tip x CapacCilx NrLocx NrUşi
Iatăcâtevainstanţeale acesteientităţi:
(Dacia, benzină, 1400, 5, 4), (Dacia, motorină, 1400, 5, 4), (Dacia, benzină, 1100, 5, 4), (Dacia, motorină, 1400, 5, 5),
(Ford, motorină, 1400, 5, 5), (Ford, benzină, 1600, 5, 4),
(Fiat, benzină, 1300, 5, 4), (Fiat, benzină, 1100, 5, 4),
(Audi, motorină, 1600, 5, 4), (Opel, benzină, 1400, 5, 5),
(Volvo, benzină, 1400, 5, 5), (Volvo, motorină, 1600, 5, 4)

Modelul relaţional
= un model formal de organizare conceptuală a datelor,

•destinat reprezentării legăturilor dintre date,
•bazat pe teoria matematică a relaţiilor,

Obiectivele modelului relaţional:
1.să permită un grad înalt de independenţă a datelor,
2.să furnizeze baze solide pentru tratarea semanticii, coerenţei şi problemelor de redundanţă a datelor,
3.să permită dezvoltarea limbajelor de prelucrare a datelor;

Modelul relaţional vs. modelele prerelationale

3***Modelele prerelaţionale***
apar două elemente:
- tipul entităţii,
- relaţiile dintre două entităţi;

accesarea BD:
- prin programe dedicate, scrise de programator,
- se face înregistrare cu înregistrare,
- se utilizează legăturile fizice între înregistrări;

Modelul relaţional
1! element: relaţia
=> orice interogare asupra BD este tot o relaţie;

independenţa modelului conceptual de implementarea fizică
=> s-au introdus o serie de limbaje neprocedurale de prelucrare a datelor .

Avantaje / dezavantaje ale modelului relaţional:
Avantaje:
•fundamentarea matematică riguroasă,
•independenţa fizică a datelor,

Limite:
•exista totuşi redundanţă,
•ocupă spaţiu,
•apar fenomene de inconsistenţă,

Regulile lui Codd
SGBD relaţional = un sistem de baze de date care respectă principiile modelului relaţional introdus de E.F. Codd;
Regula 1–regula gestionării datelor.
Regula 2–regula reprezentării informaţiei.
Regula 3–regula accesului garantat la date.
Regula 4–regula reprezentării informaţiei necunoscute.
Regula 5–regula dicţionarelor de date
Regula 6–regula limbajului de interogare.
Regula 7–regula de actualizare a vizualizării.
Regula 8–regula limbajului de nivel înalt.
Regula 9–regula independenţei fizice a datelor
Regula 10–regula independenţei logice a datelor.
Regula 11–regula independenţei datelor din punct de vedere al integrităţii.
Regula 12–regula independenţei datelor din punct de vedere al distribuirii.
Regula 13–regula versiunii procedurale a unui SGBD.
SGBDminimal relaţional:
•toate datele din cadrul bazei sunt reprezentate prin valori în tabele,
•nu există pointeri observabili de către utilizator,

•sistemul suportă operatorii relaţionali de proiecţie, selecţie şi compunere naturală, fără limitări impuse din considerente interne;

SGBDcomplet relaţional:
•este minimal relaţional şi
•sistemul suportă restricţiile de integritate de bază
•sistemul suportă toate operaţiile de bază ale algebrei relaţionale.

Principalele **caracteristici**ale modelului relaţional:
1.nu există tupluri identice,
2.ordinea liniilor şi a coloanelor este arbitrară,
3.fiecare coloană defineşte un domeniu distinct şi nu se poate repeta în cadrul aceleiaşi relaţii,
4.articolele unui domeniu sunt omogene,
5.toate valorile unui domeniu corespunzătoare tuturor cazurilor nu mai pot fi descompuse în alte valori (sunt atomice).

13. Regulile de integritate
Reguli de integritate:
•**a entităţilor**;
•**a relaţiior**(regula de **integritate referenţială**);
In plus: **restricţii contextuale(**regulile de integritate impuse de situaţia reală modelată prinbaza de date).

Regulile de integritate
(caracteristica de asigurare a integrităţii)
= aserţiuni pe care datele conţinute în BD trebuie să le satisfacă
•Clasificare:
•regulile de integritate structurale (inerente modelării datelor),
•minimale pt un SGBDR (definite în raport cu noţiunea de cheie a unei relaţii):
•de cheie,
•de entitate,
•de referinta;
•extinse
•regulile de funcţionare (de comportament: specifice unei aplicaţii particulare).

Cheiecandidat a relatiei R =
= o multime de atribute ale R ale căror valori nu sunt susceptibile de modificari si care pot identifica unic orice tuplu din R;
Cheieprimara a relatiei R =
= o mulţime minimală K de atribute ale R ale căror valori identifică unic orice tuplu din R
•i.e.: $t1, t2$tupluri ale lui R $t1(K)$ $t2(K)$ şi

K’K: $t1(K')$ $t2(K')$
•nu poate fi reactualizată ,
•reprezentare grafica: atributele componente: subliniate sau urmate de semnul „#“ ;
•R are n chei candidat şi 1! cheie primară;
•cheie primara index
•cheia primara = identifica linii in tabel (tupluri in relatie)
•indexul = localizează linii in tabel (inregistrari in fisier)

•poate folosi in acest scop o cheie secundară.

Supercheiea relatiei R =
= un grup de atribute din cadrul R care conţine o cheie a R;
□**Cheie externa**a relatiei R =

fie schemele relaţionale *R*1(*P*1, *S*1) şi *R*2(*S*1, *S*2),
unde *P*1□*R1*: cheie primară pentru *R*1,
*S*1 □*R1*: cheie secundară pentru *R*1,
*S*1 □*R2*: cheie primară pentru *R*2
□spunem că *S*1 este **cheie externă**(*foreign key*) pentru *R*1;

□Exemplu
ELEV(*CNP, CodClasă,Nume, Prenume, Adresa*)
CLASA(*CodClasă, Locaţie, nrBanci, nrTable)*
•Cheia primară poate conţine cheia externă
□Exemplu

FURNIZOR (*codF, nume, prenume, adresa*)
COMANDA (*data, codP, codF, cantitate,suma).*

Modelul relaţional respectă 3reguli de integritate structurală:
•**Regula 1**–unicitatea cheii:

cheia primară trebuie să fie unică şi minimală;
•**Regula 2**–integritatea entităţii:

atributele cheii primare trebuie să fie diferite de *null*;
•**Regula 3–**integritatea referirii:

o cheie externă trebuie să fie
-ori *null*în întregime,
-ori să corespundă unei valori a cheii primare associate

Observatie
Constrângerile de integritate pot fi implementate:
•declarativ,
•procedural (cu ajutorul declansatorilor);
□**Declanşator** =

= o procedura precompilata
stocata împreună cu BD,
invocata automat ori de câte ori are loc un anumit eveniment;

14. Proiectarea modelului relational

Problema proiectării BD:
fiind dat un volum de informaţii care trebuie reprezentat într-o BD, cum se poate alege o structură logică adecvată pentru acesta?

există câteva principii ştiinţifice care pot fi invocate:
1: Implicarea tuturor participantilor
2: Utilizarea unei abordări constructive
3: Stabilirea fazelor si activităților
4: Stabilirea unor standarde pentru dezvoltare si documentare coerenta
5: Tratarea BD ca pe nişte investitii esentiale
6: Curajul de a renunta sau de a regăndi amploarea proiectului
7: Divide et Impera
8: Proiectarea BD in vederea cresterii si modificării
•există metodologii de proiectare relativ riguroase (ex.: modelarea E/R, care are meritul că este frecvent utilizată în practică)

În practică:
•se încearcă obţinerea unei scheme conceptuale corecte

i.e. o schema logica abstracta independenta de *hardware*, SO, SGBD, limbaj, utilizator etc. ;
•se incepe cu modelarea semantică specială i.e. cu diagrama E/R;
•prezentam 9 reguli de transformare a entităţilor, relaţiilor şi atributelor acestora, în vederea obţinerii schemei conceptuale;

Notatie
In diagrama conceptuala:
•simbolul „“ indică prozitia cheii externe,
•simbolul „“ indica in plus faptul că respectiva cheie externă este conţinută în cheia primară.

A.Transformarea entităţilor
i.Entităţile independente **tabeleindependente.**
•cheia primară nu conţine chei externe
ex.:PERSONAL_FMI un tabel independent cheia primara: atributul *cnp*;
ii.Entităţile dependente **tabeledependente.**
•cheia primară a entităţilor dependente conţine cheia primară a entităţii de care depinde (cheie externă) plus unul sau mai multe atribute adiţionale
ex.:MINOR_IN_INTRETINERE un tabel dependent
•cheia primara: 2 atribute;
codP (care reprezintă cheia primară a entităţii de care depinde: PERSONAL_FMI) plus
atributul adiţional *cod_minor_intr*;
iii.Subentităţile **subtabele**
•cheia externă se referă la superentitate si coincide cu cheia primară a acesteia
ex.:CADRU_DIDACTIC subtabel
•cheia primara: *codP*este si cheia primară a entităţii PERSONAL_FMI.

B.Transformarea relatiilor Relaţiile 1:1 şi 1:*n*chei externe in tabelul secundar
ex.Fie relatia de tip 1:*n*
PICTORI_ *picteaza*_PICTURI
coloană în tabelul PICTURA:

i.e.: *cod_pictor*# PICTOR
*cod_pictor*x PICTURA
Observatie

Relaţia 1:1 plasează cheia externă în tabelul cu mai puţine linii.

Relaţia *m*:*n*□**tabelasociativ**
•cheia primara = juxtapunerea celor doua chei primare (a tabelului principal si a tabelului secundar, devenite chei externe in tabelul asociativ), eventuale coloane adiţionale;
•tabelele asociative se desenează punctat;
□ex.: relaţia de tip*n:m*
*STUDENTI_urmeaza_CURS*URI
□tabel asociativ:
i.e.: (*codS, codC*)# □URMEAZA
codS#□STUDENT
codC#□CURS.

Relaţiile de tip trei □tabele asociative
•cheia primara = juxtapunerea celor trei chei primare (devenite chei externe in tabelul asociativ), eventuale coloane adiţionale;
□ex.: relaţia de tip 3 *activitate_didatica*stabilita intre entităţile CADRU_DIDACTIC, STUDENT şi CURS
□tabel asociativ:
i.e.: (*codP, codS, codC*)# □ACTIVITATE_DIDACTICA
codP#□CADRU_DIDACTIC
codS#□STUDENT
codC#□CURS
□Observatie
Entitatea CADRU_DIDACTIC are cheia primara *codP*intrucat este subentitate a entitatii PERSONAL_FMI.
In acest caz, este preferabilă o cheie primară artificială.

Transformarea atributelor
□Un atribut singular □o coloană;
□Atributele multiple □tabele dependendente ce conţin:
•cheia primară a entităţii,
•atributul multiplu
•cheia primară: este formată din:
•o cheie externă,
•una sau mai multe coloane adiţionale;
□ex.un angajat al FMI poate avea mai multe adrese email
=> atributul *adresa_email* este atribut multiplu
□tabelul dependent ADRESA_EMAIL
□Atributele entităţilor □coloane in tabelele corespunzatoare,
□Atributele relaţiilor:
•relaţii 1:1 şi 1:*n* □coloane in tabelul care conţine cheia externă,
•relaţii *m*:*n*şi de tipul trei □coloane in tabelele asociative.

Transformarile de mai sus genereaza urmatoarea clasificare a tabelelor dintr-o BD (in functie de structura cheii primare):

Tabel	***reprezintă***	***Cheie primară***
independent	**entitate independentă**	nu conţine chei externe

subtabel	**subentitate**	o cheie externă
dependent	**entitate dependenta**	o cheie externă şi una sau mai multe coloane adiţionale
	atribut multiplu	
asociativ	**relaţie *m:n***	
	relaţie de tip 3	două sau mai multe chei externe şi (opţional) coloane adiţionale

15. Operatorii algebrei relationale (clasificari, definitii, exemple)

1.SELECT(selecţie) –extrage tupluri ce satisfac o condiţie specificată;
2.PROJECT(proiecţie) –extrage atributele specificate;
3.DIFFERENCE(diferenţă) –extrage tupluri care apar într-o relaţie, dar nu apar în cealaltă;
4.PRODUCT(produs cartezian) –generează toate perechile posibile de tupluri, primul element al perechii fiind luat din prima relaţie, iar cel de-al doilea element din cealaltă relaţie;
5.UNION(reuniune) –reuneşte două relaţii;
6.INTERSECT(intersecţie) –extrage tupluri care apar în ambele relaţii;
7.DIVISION(diviziune) –extrage valorile atributelor dintr-o relaţie, care apar în toate valorile atributelor din cealaltă relaţie;
8.JOIN(compunere) –extrage tupluri din mai multe relaţii corelate:
9.NATURAL JOIN (compunere naturală) –combină tupluri din două relaţii, cu condiţia ca atributele comune să aibă valori identice;
10.SEMI-JOIN(semi-compunere) –selectează tupluri doar dintr-o relaţie care vor fi corelate cu tuplurile celeilalte relaţii;
11.Θ-JOIN(Θ-compunere) –combină tupluri din două relaţii, cu condiţia ca valorile atributelor specificate să satisfacă o anumită condiţie;
12.OUTER JOIN (compunere externă) –combină tupluri din două relaţii, astfel încât condiţiile de corelare să fie satisfăcute. Tuplurile din orice relaţie care nu satisfac aceste condiţii sunt completate cu *null*.

Observatii

1.UNION, INTERSECT, DIFFERENCE:
•se aplica numai la relaţii având aceeaşi aritate,
•ordinea (nu numele) atributelor este aceeaşi;
2.Scopul fundamental al **AR**: scrierea expresiilor relaţionale;

Aplicaţii posibile ale expresiilor relaţionale:
•definirea unui domeniu pentru interogare sau actualizare,
•definirea constrângerilor de integritate şi de securitate,
•definirea datelor care vor fi incluse într-o vizualizare,

•definirea datelor care vor reprezenta domeniul de valabilitate al unei operaţii de control al concurenţei etc

Operatorul *PROJECT*

Proiecţia =

= o operaţie unară care elimină anumite atribute ale unei relaţii *R*, producând o submulţime „pe verticală“ a acesteia

Observatie

Suprimarea unor atribute poate avea ca efect apariţia unor tupluri duplicate, care trebuie eliminate

Notatii

*ΠA*1, ..., *Am*(*R*),
PROJECT(*R*, *A*1, ..., *Am*),
R[*A*1, ..., *Am*],
unde *A*1, *A*2, ..., *Am*sunt parametrii proiecţiei relativ la relaţia *R*

*A*1, *A*2, ..., *Am*sunt atributele din *R*care nu au fost elimiate prin proiectie (care apar in relatia-rezultat *R'*).

Exemplu:

Să se obţină numele si prenumele salariatilor din FMI
Proiecţie în algebra relaţională:
Rezultat = PROJECT (PERSONAL_FMI, nume, prenume)
Proiecţie fără dubluri în *SQL*:
SELECTDISTINCT nume, prenume
*FROM*personal_fmi;

Operatorul *SELECT*

Selectia =

= o operaţie unară care elimină anumite tupluri ale unei relaţii *R*, producând o submulţime „pe orizontala“ a acesteia

Observatii

Relatia *R'* rezultata din relatia *R* se obţine prin extragerea tuplurilor din *R*care satisfac o condiţie specificată

Condiţia este o formulă logică ce poate cuprinde nume de atribute, constante, operatori logici, operatori aritmetici de comparare;

Notatii

SELECT (R, condiţie),
R [condiţie],
RESTRICT (R, condiţie)

Exemplu:

Să se obţină toate informatiile despre cursurile optionale din FMI
Selectie în algebra relaţională:
Rezultat = SELECT (CURS, tip=„optional“)
Selectie în *SQL*:
*SELECT**
*FROM*curs
*WHERE*tip=„optional“;

Operatorul *UNION*

Reuniunea =

= fie *S*si *T*doua relatii de aceeasi aritate; reuniunea lor, *R*, este tot o relatie, care consta din multimea tuplurilor aparţinând fie lui *S*, fie lui *T*, fie ambelor relaţii

Observatii
Reuniunea este o operatie binara comutativa;
Operatorul de reuniune permite:
•obţinerea tuplurilor distincte a două relaţii
•adăugarea de noi tupluri într-o relaţie;

Notatii
UNION (S, T)
OR (S, T)
APPEND (S, T)
Exemplu:
Să se obţină lista completa a numelor si prenumelor cadrelor didactice si studentilor din FMI
Reuniune în algebra relaţională:
S = PROJECT (CADRU_DIDACTIC, nume, prenume)
T = PROJECT (STUDENT, nume, prenume)
Rezultat = UNION (S,T)
Reuniune în *SQL*:
*SELECT*nume, prenume
*FROM*cadru_didactic
UNION
*SELECT*nume, prenume
*FROM*student;
Operatorul *DIFFERENCE*
Diferenta =
= fie *S*si *T*doua relatii de aceeasi aritate; diferenta lor, *R*, este tot o relatie, care consta din multimea tuplurilor care aparţin lui *S*, dar nu aparţin lui *T*
Observatii
Diferenta este o operatie binara NEcomutativa;
Operatorul de diferenta permite:
•obţinerea tuplurilor ce apar numai într-o relaţie
•stergerea tuplurilor dintr-o relaţie;
Notatii
DIFFERENCE (*S, T*)
MINUS (*S, T*)
REMOVE (*S, T*)
S–T.
Exemplu:
Să se obţină numele studentilor care nu se regăsesc printre numele angajatilor din FMI
Diferenta în algebra relaţională:
S = PROJECT (STUDENT, nume)
T = PROJECT (PERSONAL_FMI, nume)
Rezultat = DIFFERENCE (S,T)
Diferenta în *SQL*:
*SELECT*nume
*FROM*student
MINUS
*SELECT*nume
FROM personal_fmi;
Operatorul *INTERSECT*

Intersectia =
= fie *S*si *T*doua relatii de aceeasi aritate; intersectia lor, *R*, este tot o relatie, care consta din multimea tuplurilor care aparţin atat lui *S*cat si lui *T*
Observatii
Intersectia este o operatie binara comutativa;
Operatorul de intersectie permite:
•obţinerea tuplurilor ce apar simultan in doua relatii
Operatorul INTERSECT este un operator derivat:
$S \cap T = S - (S - T)$
$S \cap T = T - (T - S)$.
Notatii
INTERSECT (S, T)
AND (S, T)
Exemplu:
Să se obţină numele studentilor care coincid cu numele angajatilor din FMI
Intersectia în algebra relaţională:
S = PROJECT (STUDENT, nume)
T = PROJECT (PERSONAL_FMI, nume)
Rezultat = INTERSECT (S,T)
Intersectia în *SQL*:
*SELECT*nume
*FROM*student
INTERSECT
*SELECT*nume
FROM personal_fmi
Operatorul *PRODUCT*
Produsul cartezian =
= fie *S*si *T*doua relatii de aritate *m*, respectiv *n*; produsul cartezian al lui *S*cu *T*este tot o relatie, fie ea *R*, care consta din multimea tuplurilor de aritate *m*+*n*cu proprietatea ca, in fiecare tuplu, primele *m*componente reprezinta un tuplu din *S*iar celelalte *n*componente reprezinta un tuplu din *T*
Observatii
Produsul cartezian este o operatie binara NEcomutativa;
Este posibil ca cele două relaţii să aibă atribute cu acelaşi nume

=> pentru a menţine unicitatea denumirilor atributelor din cadrul unei relaţii, denumirile acestor atribute se prefixeaza cu denumirea relaţiei.
Notatii
PRODUCT(*S*, *T*)
TIMES (*S*, *T*)
Exemplu:
Să se obţină lista tuturor posibilitatilor de alocare de cursuri cadrelor didactice din FMI
Produs cartezian în algebra relaţională:
Rezultat = PRODUCT (CADRU_DIDACTIC, CURS)
Produs cartezian în *SQL*:
*SELECT**
*FROM*cadru_didactic, curs;

Operatorul *DIVISION*
Diviziunea=

= fie doua multimi de atribute:
A = {*A1, A2, ... , An*}şi
B = {*B1, B2, ... , Bm*} si
doua relatii *S(A,B),* de aritate *n+m*si *T(B)* de aritate *m*
(i.e.:multimea atributelor relatiei *T*este o submultime a multimii atributelor relatiei *S*);
=> rezultatul aplicarii operatorului de diviziune asupra relatiilor *S*si *T*este o relatie *R*de aritate *n*cu proprietatea ca:
multimea atributelor sale coincide cu multimea de atribute *A* (i.e.: consta din acele atribute care apartin relatiei *S*si nu apartin relatiei *T*),
multimea tuplurilor sale rezulta prin selectarea tuplurilor din relatia S astfel: fie r[A]□R => □t[B]□T a.i s[A,B]□S
Operatorul *DIVISION* (cont.)
R= S□T = □A□S.B=T.B(S)
adica: intai se selecteaza un tuplu din S doar daca valorile atributelor sale B coinid cu valorile atributelor unui tuplu din T
apoi, pe un astfel de tuplu,se aplica o proiectie pt a retine doar valorile atributelor din A;
acestea formeaza un tuplu din R;

Notatii
DIVIDE (S, T)
DIVISION (S, T)
S□T
□Division = operator derivat:
R= S□T = □A□S.B=T.B(S) sau:
*S□T= S*1–*S*2.
unde: *S*1= Π1,2,...,*n* (*S*), *S*2= Π1,2,...,*n*((*S*1□*T*) –*S*)
□Division = exprimare in SQL:

necesita utilizarea □(care nu exista in *SQL*!)
dar □poate fi simulat cu ajutorul □stiind ca:
□*xP*(*x*) □¬ □*x*¬ *P*(*x*)
=> operatorul DIVISION poate fi exprimat în *SQL*prin succesiunea a doi operatori *NOT EXISTS.*
Exemplu:
Să se obţină codurile studentilor care urmeaza cel putin un curs facultativ
Diviziune în algebra relaţională:
S= PROJECT (URMEAZA, codS, codC)
T= PROJECT (SELECT (CURS, tip=„facultativ“), codC)
Rezultat = DIVISION (*S,T*).

Diviziune în *SQL:*
*SELECTUNIQUE*codS
*FROM*urmeaza x
WHERENOTEXISTS
(*SELECT**
*FROM*curs c
*WHERE*curs.tip = „facultativ“
ANDNOTEXISTS
(*SELECT**
FROM urmeaza b

*WHERE*c.codC = b.codC
*AND*b.codS = x.codS));

Operatorul *JOIN*
= operator de **compunere**care permite regăsirea informaţiei din mai multe relaţii corelate
Compunerea=

= operaţie binară asupra a 2 relatii, *S, T*, care are ca rezultat o nouă relaţie, *R*, în care fiecare tuplu este o combinaţie a unui tuplu din *S*cu un tuplu din *T*
Observatii
1.Condiţia necesară aplicării operatorului JOIN:

tuplurile care se combină să fie similare
2.Operatorul combina alti 3 operatori:
•produsul cartezian,
•selecţia,
•proiecţia;

În general:
•se construieşte un produs cartezian,
•se elimină tupluri prin selecţie,
•se elimină atribute prin proiecţie;
3.Există mai multe variante ale operatorului *JOIN*:
•*NATURAL JOIN*,
•*Θ-JOIN*
•*SEMI-JOIN*,
•*OUTER JOIN* (*LEFT, RIGHT, FULL*).

Operatorul *NATURAL JOIN*
Operatorul de compunere naturala =
= combină tupluri din două relaţii *S*şi *T*, cu condiţia ca atributele comune să aibă valori identice
Notatii
JOIN(S, T)
R◊S
Algoritmul
1.se calculează produsul cartezian *S T*,
2.pentru fiecareatribut comun *A*care defineşte o coloană în *S*şi o coloană în *T:*

se selectează din *S T* tuplurile ale căror valori coincid în coloanele *S.A*şi *T.A*(atributul *S.A*reprezintă numele coloanei din *S T* corespunzătoare coloanei *A*din *S*),
3.pentru fiecare astfel de atribut *A*se proiectează coloana *T.A*, iar coloana *S.A*se va numi *A*;
NATURAL JOIN = operator derivat:
JOIN (*S, T*) = Π*i*1,...,i*m*σ(*S.A*1 = *T.A*1) ... (*S.Ak*= *T.Ak*)(*S T*),
unde *A*1, ..., *Ak*= atributele comune lui *S*şi *T*,
*i*1, ..., *im*= lista componentelor din *S T*(păstrând ordinea iniţială) din care au fost eliminate componentele *S.A*1, ..., *S.Ak*
Exemplu:
Să se obţină informaţii complete despre studentii FMI si liceele pe care le-au absolvit, respectiv.
NATURAL JOIN în algebra relaţională:

Rezultat = JOIN (STUDENT, LICEU)
NATURAL JOIN în *SQL*:
SELECT *
*FROM*student s, liceu l
WHERE s.codL = l.codL;

Operatorul *θ-JOIN*
Operatorul de *θ*-compunere=
= combină tupluri din două relaţii *S*şi *T*, cu condiţia ca atributele mentionate sa indeplineasca o anumita conditie specificata explicit in cadrul operatiei
Notatii

JOIN(*R, S*, conditie)
θ-JOIN= operator derivat: produs cartezian si selectie
JOIN (*S, T, conditie*) = *σ*conditie (*S T*)

Exemplu:
Să se obţină informaţii despre studentii FMI (cod, nume, prenume, data şi locul nasterii) şi despre cadrele didactice (cod, nume, prenume, grad didactic, doctorat) cu condiţia ca numele de familie ale cadrelor didactice şi studentilor sa nu coincida
Operatorul ***θ-JOIN***în algebra relaţională:
S = PROJECT (STUDENT, codS, nume, prenume, data_nastere, loc_nastere)
T = PROJECT (CADRU_DIDACTIC, codCD, nume, prenume, gradD, doctorat)
Rezultat = JOIN (S; T, STUDENT.nume <> CADRU_DIDACTIC.nume)
Operatorul ***θ-JOIN***în *SQL*:
*SELECT*codS, nume, prenume, data_nastere, loc_nastere, codCD, nume, prenume, gradD, doctorat
*FROM*student s, cadru_didactic c
*WHERE*s.nume <> c.nume;

Operatorul *SEMI-JOIN*
Operatorul de semi-compunere=
= generează o relaţie care conţine toate tuplurile din *S*ce sunt corelate cu cel puţin unul dintre tuplurile din *T*
Observatii
1.Operatorul este utilizat când nu sunt necesare toate atributele compunerii (sunt conservate atributele unei singure relaţii participante la compunere),
2.Operatorul este asimetric;
Notatii

SEMIJOIN(S, T)
SEMIJOIN(S, T, condiţie).
SEMI-JOIN= operator derivat:
SEMIJOIN (*S, T*) = Π*M*(*JOIN* (*S, T*))
SEMIJOIN (*S, T*, condiţie) = Π*M*(*JOIN* (*S, T*, condiţie)),
unde *M*= mulţimea atributelor relaţiei *S*.

Exemplu:
Să se obţină informaţii (nume, localitate) despre liceele ai caror absolventi de alta nationalitate decat cea romana au devenit studenti ai FMI

Operatorul *SEMI-JOIN*în algebra relaţională:
S = SELECT (STUDENT, nationalitate <>„romana“)
T = JOIN (S, LICEU)
Rezultat = PROJECT (T, denumire, oras)
Operatorul *SEMI-JOIN*în *SQL*:
*SELECT*denumire, oras
*FROM*student s, liceu l
*WHERE*s.codL= l.codL
*AND*nationalitate <>“romana”

Operatorul *OUTER-JOIN*
Operatorul de compunere externa=
= combină tuplurile din două relaţii *S*si *T*pentru care sunt satisfăcute condiţiile de corelare fără a pierde, însă, celelalte tupluri
Observatii
1.În cazul aplicării operatorului *JOIN*se pot pierde tupluri (există un tuplu în una din relaţii pentru care nu există niciun tuplu în cealaltă relaţie, astfel încât să fie satisfăcută relaţia de corelare)
2.Operatorul *OUTER JOIN* elimină acest inconvenient astfel:
•practic, se realizează compunerea naturala a două relaţii *S*şi *T*
•apoi se adaugă tuplurile din *S*şi *T*, care nu sunt conţinute în compunere, completate cu *null*acolo unde valorile atributelor există într-un tuplu din *S*(respectiv *T*) dar nu există şi în *T*(respectiv *S*);
avantajul acestui operator:
se păstrează informaţiile (i.e.: se păstrează tuplurile care ar fi fost pierdute în alte tipuri de *join*).
Notatii
OUTERJOIN(S, T)
OUTERJOIN(S, T, condiţie).
În practică apar trei tipuri de operatori OUTER JOIN:
•LEFT OUTER JOIN păstrează în rezultat fiecare tuplu al relaţiei din stânga (aici: *S*)
•RIGHT OUTER JOIN păstrează în rezultat fiecare tuplu al relaţiei din dreapta (aici: *T*)
•FULL OUTER JOIN păstrează tuplurile din ambele relaţii, completate cu *null*atunci când nu există tupluri corelate.
Exemplu:
Să se obţină informaţii complete referitoare la studentii FMI şi la cursurile optionale urmate de aceştia, sesizând cazurile în care există cursuri optionale la care nu s-a inscris niciun student, respectiv studenti care nu s-au inscris la niciun curs optional
Operatorul *OUTER JOIN* în algebra relaţională:
Rezultat = OUTERJOIN (STUDENT, CURS)
Operatorul *OUTER JOIN* în *SQL*:
*SELECT**
*FROM*student s *FULLOUTERJOIN*
(SELECT *
FROM curs c
HERE c.tip =“optional“)

16. Proprietatile operatorilor relationali
Proprietatile operatorilor AR
Expresie a algebrei relaţionale =

o expresie in care:

•operanzii = relatii (in sensul lui E.F. Codd),
•operatorii =
•cei 8 operatori (primitivi sau derivati) ai **AR**(proiectie, selectie, etc.), plus, eventual,
•operatorii suplimentari (complement, despicare , inchidere tranzitiva) si
•functiile asociate (MIN, MAX, AVG, VAR);
□Două expresii sunt **echivalente**□
în urma evaluării lor, se obţine ca rezultat aceeaşi relaţie.

Observatii
•O expresie **AR**□un plan de executie a cererii;
•O expresie se poate reprezenta grafic cu ajutorul unui arbore, numit **arbore algebric**, în care nodurile corespund operatorilor din cadrul expresiei respective

•Evaluarea unei expresii:
efectuarea prelucrărilor indicate de operatori
•în ordinea apariţiilor acestora, sau
•în ordinea fixată prin paranteze;
•Rezultatul evaluării unei expresii:

o relaţie derivată din relaţiile menţionate ca operanzi în cadrul expresiei;
•Ordinea în care se efectuează operaţiile:
rolul cel mai important în evaluarea costului necesar realizării interogării.
2 metode de determinare a ordinii optime de execuţie a operaţiilor dintr-o expresie **AR**:
1.algebric,
2.prin estimarea costului:

(1) Optimizarea cererilor bazată pe **AR** :
•se exprimă cererile sub forma unor expresii algebrice relaţionale,
•se aplică transformări algebrice care conduc la expresii echivalente, dar care vor fi executate mai eficient;
aceste transformari au la baza o strategie de optimizare:
•independentă de modul de memorare a datelor (strategie generală),
•dependentă de modul de memorare (strategie specifică unui anumit SGBD);

□Regulile de transformare a unui plan de executie / expresie **AR**□proprietăţi ale operatorilor algebrici care permit ordonarea într-o altă formă, mai convenabilă, a operaţiilor din interogare.

Proprietatea 1. Operaţiile *join*şi produs cartezian: comutative
$JOIN(R1, R2) = JOIN(R2, R1)$
$R1 \square R2 = R2 \square R1$
Proprietatea 2. Operaţiile *join*şi produs cartezian: asociative
$JOIN(JOIN(R1, R2), R3) = JOIN(R1, JOIN(R2, R3))$
$(R1 \square R2) \square R3 = R1 \square (R2 \square R3)$
Proprietatea 3. Compunerea proiecţiilor:
$\Pi A1,...,Am(\Pi B1,...,Bn(R)) = \Pi A1,...,Am(R)$
unde $\{A1, A2,...,Am\} \square \{B1, B2,...,Bn\}$
Proprietatea 4. Compunerea selecţiilor:
$\sigma cond1(\sigma cond2(R)) = \sigma cond1 \square cond2(R) = \sigma cond2(\sigma cond1(R))$
unde am notat prin *cond*condiţia după care se face selecţia

Proprietatea 5. Comutarea selecţiei cu proiecţia:
dacă *cond*implică numai atributele *A*1,...,*Am* atunci:
Π*A*1,...,*Am*(σ*cond*(*R*)) = σ*cond*(Π*A*1,...,*Am*(*R*))
dacă *cond*implică şi atributele *B*1,...,*Bn*, care nu aparţin mulţimii {*A*1,...,*Am*}
Π*A*1,...,*Am*(σ*cond*(*R*)) = Π*A*1,...,*Am*(σ*cond*(Π*A*1,...,*Am*,*B*1,...,*Bn*(*R*)))
Proprietatea 6. Comutarea selecţiei cu produsul cartezian:
dacă *cond*implică numai atribute ale relaţiei *R*1 atunci:
σ*cond*(*R*1 □*R*2) = σ*cond*(*R*1) □*R*2
dacă *cond*= *cond*1□*cond*2
*cond*1 implică numai atribute din *R*1
*cond*2 implică numai atribute din *R*2, atunci:
σ*cond*(*R*1 □*R*2) = σ*cond*1(*R*1) □σ*cond*2(*R*2)
dacă *cond*1 implică numai atribute din *R*1
*cond*2 implică atribute atât din *R*1 cât şi din *R*2, atunci:
σ*cond*(*R*1 □*R*2) = σ*cond*2(σ*cond*1(*R*1) □*R*2) .
Proprietatea 7. Comutarea selecţiei cu reuniunea:
σ*cond*(*R*1 □*R*2) = σ*cond*(R1) □σ*cond*(*R*2)
Proprietatea 8. Comutarea selecţiei cu diferenţa:
σ*cond*(*R*1 –*R*2) = σ*cond*(*R*1) –σ*cond*(*R*2)
Proprietatea 9. Comutarea proiecţiei cu reuniunea:
Π*A*1,...,*Am*(*R*1 □*R*2) = Π*A*1,...,*Am*(*R*1) □Π*A*1,...,*Am*(*R*2)
Proprietatea 10. Comutarea proiecţiei cu produsul cartezian:
dacă {*A*1,...,*Am*} = {*B*1,...,*Bn*,*C*1,...,*Ck*}
unde *B*1,...,*Bn* sunt atribute ale relatiei *R*1
*C*1,...,C*k* sunt atribute ale relatiei *R*2 atunci:
Π*A*1,...,*Am*(*R*1 □*R*2) = Π*B*1,...,*Bn*(*R*1) □Π*C*1,...,*Ck*(*R*2)
Proprietatea 11. Compunerea proiecţiei cu operaţia *join*:
dacă {*A*1,...,*Am*} = {*B*1,...,*Bn*,*C*1,...,*Ck*}
unde *B*1,...,*Bn* sunt atribute ale relatiei *R*1
*C*1,...,C*k* sunt atribute ale relatiei *R*2 atunci:
Π*A*1,...,*Am*(*JOIN*(*R*1,*R*2,*D*)) =
Π*A*1,...,*Am*(*JOIN*(Π*D*,*B*1,...,*Bn*(*R*1), Π*D*,*C*1,...,*Ck*(*R*2),*D*),
unde am notat prin *JOIN*(*R*1, *R*2, *D*) operaţia de compunere naturală între *R*1 şi *R*2 după atributul comun *D*.
Proprietatea 12. Compunerea selecţiei cu operaţia *join*:
σcond(*JOIN*(*R*1, *R*2, *D*)) =
σcond(*JOIN*(Π*D*,*A*(*R*1), Π*D*,*A*(*R*2), *D*)),
unde *A*reprezintă multimea de atribute care apar în *cond*

17. Optimizarea interogarilor

Optimizarea interogarilor
Obiectivul optimizarii:
creşterea vitezei de access la informaţia din BD □
□reducerea timpului de acces
□Evident:interogarile care necesita cel mai lung timpde prelucrare: interogarile care contin operatorii:
•produs cartezian,
•compunere;

Cele mai utilizate metode :

1.reducerea timpului de răspuns (timpul total de execuţie a interogării suma timpilor de execuţie pentru fiecare dintre operaţiile elementare care compun interogarea):
•prin minimizarea timpului de executie al fiecarei operatii elementare care compune introgarea,
•prin maximizarea numărului de operaţii paralele;
2.metoda euristica:
•are in vedere ordinea de executie a operatiilor **AR**,
•se bazeaza pe cele 12 proprietati enumerate mai sus.

Ambele metode depind de informaţiile statistice despre BD
(ex.: în dicţionarul datelor pot fi stocate informaţii statistice referitoare la:
•cardinalitatea relaţiilor,
•numărul de blocuri necesare pentru stocarea unei relaţii,
•numărul de tupluri dintr-o relaţie care intră într-un singur bloc,
•numărul de niveluri dintr-un index,
•numărul de valori distincte pentru fiecare atribut etc.)

=> reactualizarea: cand sistemul este cu activitate redusă, şi nu de fiecare dată când este inserat, şters sau reactualizat un tuplu.

Examinarea celor 12 proprietati:
conturarea strategiei de optimizare (de reordonare a operatiilor din cereri):
•efectuarea mai întâi a operaţiilor unare (selectii inainte de proiectii) -care reduc dimensiunea relaţiilor - şi apoi a operaţiilor binare (=> coborârea selecţiilor şi proiecţiilor cât mai jos posibil în arborele algebric),
•regruparea compunerilor de selecţii şi proiecţii într-o selecţie urmată de o proiecţie,
•regruparea selecţiilor şi proiecţiilor "in cascadă", prin unul dintre operatorii algebrici binari,
•combinarea proiecţiilor cu operaţiuni binare adiacente,
•combinarea unora din selecţii cu produse carteziene care eventual le preced, pentru a obţine compuneri,
•înainte de compuneri, prelucrareapreliminarăafişierelor (relaţiilor) prin operaţiuni de sortare şi de indexare,
•căutarea în expresiilemai complexe asub-expresiilorcare se repetă.

Formal: 4 reguli euristice de optimizare:
Regula de optimizare 1.Selecţiile se execută cât mai devreme posibil pt ca reduc substanţial dimensiunea relaţiilor;
Proprietatea 4 poate fi folosită pentru a separa două sau mai multe selecţii în selecţii individuale :

$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond1 \wedge cond2}(R) = \sigma_{cond2}(\sigma_{cond1}(R))$
aceste selectii individuale pot fi distribuite *join*-ului sau produsului cartezian folosind comutarea selecţiei cu *join*-ul
Regula de optimizare 2.Produsele carteziene se înlocuiesc cu *join*-uri, ori de câte ori este posibil
un produs cartezian între două relaţii generează toate combinatiile de tupluri din cele 2 relatii (i.e. un cardinal f mare)
această transformare se poate realiza folosind legătura dintre produs cartezian, *join*şi selecţie.

Regula de optimizare 3.Dacă sunt mai multe *join*-uri atunci cel care se execută primul este cel mai restrictiv
un *join*este mai restrictiv decât altul dacă produce o relaţie mai mică
se poate determina care *join*este mai restrictiv:

•pe baza factorului de selectivitate
•cu ajutorul informaţiilor statistice
•cu Proprietatea 2: asociativitatea operaţiei de *join:*

JOIN(*JOIN*(*R*1, *R*2), *R*3) = *JOIN*(*R*1, *JOIN*(*R*2, *R*3))
Regula de optimizare 4.Proiecţiile se execută la început pentru a îndepărta atributele nefolositoare.

Algoritm pentru optimizarea expresiilor relationale
Intrare:o expresie relationala, reprezentata printr-un arbore sintactic
Ieşire: o expresie relationala optimizata, reprezentata tot printr-un arbore sintactic
Metoda:
Pas 1. Fiecareselectie σ*cond*1□*cond*2□... □*cond*n(*R*) este transformată în secvenţaσ*cond*1(σ*cond*2(...(σ*cond*n(*R*)))) *(Proprietatea* 4)
Pas 2. Fiecare selecţie este deplasată cât mai jos posibil în arborele sintactic (*Proprietatile* 4-7)
Pas 3. Fiecare proiecţie este deplasată cât mai jos posibil în arborele sintactic (*Proprietatile* 3, 10, 9, 5)
Pas 4. Secvenţele de selecţii şi proiecţii sunt combinate în:
•selecţii unice,
•proiecţii unice, sau
•selecţii urmate de proiecţii (regulile 3, 4, 5)

□Atentie:operatiile Pasului4, pot încălca principiul potrivit căruia proiecţiile sunt efectuate cât mai curând posibil
□trebuie analizatcastigul fiecarei variante: Pasul 4 sau Propr. 5!

Algoritm pentru optimizarea expresiilor relationale(cont.)
Pas 5. Nodurile interne ale arborelui rezultate prin parcurgerea paşilor anteriori sunt grupate în „blocuri”
•Fiecare nod intern care corespunde unei operaţiuni binare poate face parte din acelaşi bloc ca şi predecesorii săi imediaţi cu care sunt asociate operaţiuni unare
•Din bloc poateface parte şi orice lanţ de noduri succesoare asociate cu operaţiuni unare şi terminate cu o frunză

□Ultima regulă nu se aplică atunci când operaţia binară este un produs carteziannеurmat de o selecţie caresase combine cu produsul menţionat, astfel încât să formeze o θ-compunere!
Pas 6. Se evaluează fiecare bloc, în orice ordine, astfel încât niciunul din blocuri nu este evaluat înaintea grupurilor sale succesoare.

Exemplu.
Fie o baza de date cu entitatile:
CIRCUIT (*Cnume, Fnume, Cod*),
FURNIZOR (*Fnume, Fadr*),
UTILIZATOR (*Unume, Uadr, Nrdoc*),
LIVRARI (*Nrdoc*, *Cod*, *Data*);
Ppcă pentru a returna anumite informatii privind livrările de circuite este construită mai întâi o vizualizare care conţine date referitoare la circuitele livrate
□se utilizeaza relaţiileLIVRARI, UTILIZATOR, CIRCUIT;

Vizualizarea va fi definită prin expresia relaţională:
ΠV(□U(LIVRARIx UTILIZATORx CIRCUIT)),
unde:

V = {*Cnume, Fnume, Cod, Unume, Uadr, Nrdoc, Data*}
U = UTILIZATOR.Nrdoc = LIVRARI.*Nrdoc*∧CIRCUIT.Cod
= LIVRARI.Cod.

Pp că interogareasolicită lista numelor circuitelor livrate înainte de 14februarie 2014
□În termenii algebrei relaţionale:

ΠCnume(σData<14.02.2014(ΠU(σV(LIVRARIxUTILIZATORxCIRCUIT))))
Pentru evaluarea expresiei: se construieste arborele ei sintactic
se aplica algoritmul de optimizare

18. Anomalii în proiectarea modelelui relaţional

Reamintim:
Tipuri de reguli integritate:
•a entităţilor:
•a relaţiior
•restricţii contextuale

Pentru a prezenta procesul de normalizare, este necesar să definim următoarele două concepte:
1.anomalie,
2.dependenţă funcţională.

□Clasificare
I. Redundanţă logică
II. Anomaliile la actualizare
II.a. anomalie la inserţie
II.b. anomalie la ştergere
II.c. anomalie la modificare
III. Problema reconexiunii (corelata, in general, cu operatiile de compunere).

19. Dependenţe functionale în BD relationale
Dependenţă funcţională =
= o restricţie care apare între atributele unei entităţi la nivelul semanticii (semnificaţiei) acestora şi al valorilor lor:
fie a1şi a2atributele unei entităţi E; spunem că atributul **a2** este **dependent funcţional** de atributul **a1** (sau: atributul **a1 determina functional** atributul **a2**)
□pentru fiecare valoare a atributului a1exista cel mult o valoare a atributului a2 ;
□atunci cand mai multe tupluri ale entitatii E iau aceleasi valoare pentru atributul a1ele iau valoare şi pentru atributul a2
□Notatie

a1→a2
Determinantul unei dependenţe funcţionale =
= atributul care, prin valorile sale, determină valorile celuilalt atribut
(i.e.: atributul aflat, în ambele reprezentări, în stânga săgeţii);
□Atributul **a1** se numeste **determinant**si atributul **a2**se numeste **determinat**al unei dependente functionale
□pentru fiecare valoare a atributului **a1**există cel mult o valoare a atributului **a2**

•Examinarea dependenţelor funcţionale dintre atributele unei relatii

□determinarea cheilor candidat precum şi a cheii candidat care trebuie să fie aleasă drept cheie primară: este aleasă cheia candidat care apare ca determinant în toate dependenţele funcţionale identificate la nivelul entităţii respective**.**

Un algoritm de calcul al inchiderii unui set de atribute aflate in dependenta functionala
□Definitie
Fie U o relatie [universala],
X = {A1, A2, ... , An} o multime de atribute ale U,
S = o multime de dependente functionale (considerate de proiectantul BD);
se numeste **inchidere a multimii de atribute X determinata de multimea de dependente functionale S**, acea multime de atribute Y cu proprietatea ca orice relatie care satisface toate dependentele din S satisface si dependenta A1A2...An □Y
(i.e. dependentele din S implica "automat" dependenta A1A2...An□Y);
□Notatie

X+
{A1, A2, ... , An}+.

Algoritm de calcul pentru inchiderea X+a unui set de atribute X in raport cu un set de depenedente functionale S
1.Se initializeaza X+cu multimea de atribute X = {A1, A2, ... , An} implicate in dependentele functionale din multimea S
2.Se cauta o noua dependenta functionala B1B2...Bk□C cu proprietatea ca: □1□i□k, k□n: Bi□A dar C□A
3.X+= X□{C}
4.Se reiau Pasul 2 si Pasul 3 pana cand nu se mai pot adauga noi atribute la X
5.Multimea X+astfel obtinuta constituie inchiderea multimii de atribute X in raport cu S;

□Observatie
Algoritmul produce, corect, inchiderea X+a multimii X pentru ca:
(i)multimea de atribute ale oricarei relatii [universale] este finita,
(ii)se adauga –eventual –atribute care fac parte numai din respectiva relatie,
(iii)la niciun pas din algoritm, dimensiunea multimii X+nu se micsoreaza.

Observatie
Fie R o relatie, X multimea tuturor atributelor sale şi A □X =>
A+= X □A este o supercheie pentru R
(i.e. A determina functional toate atributele lui R atunci şi numai atunci cand A+coincide cu multimea tuturor atributelor din R)
=> putem cauta / verifica daca o multime de atribute A este o supercheie pentru relatia R astfel:
(i) aplicam multimii A algoritmul de mai sus şi calculam inchiderea A+;
(ii)testam daca A+= X;
(iii)daca da, testam pentru □B □A daca B+= X;
(iv)daca nu exista nicio multime B □A a.i. B+= X, atunci algoritmul se incheie cu A = supercheie pentru R;
(v)daca □B' □A a.i. B'+= X, atunci inlocuim pe A cu B' şi reluam Pasul (i).

Definitii(tipuri de dependente)
(1)O dependenţă funcţională $X \square Y$se numeşte **dependenţă funcţională totală**(FT)

□nu există nicio submulţime proprie $X \square \square X$a. î. $X \square \square Y$
(2)O dependenţă funcţională $X \square Y$se numeşte **dependenţă funcţională partiala**

□dacă există o submulţime proprie $X \square \square X$a. î. $X \square \square Y$
(3) Fie *D* mulţimea dependenţelor unei relaţii şi
*p*1, *p*2, ..., *pr*, *r*□1, proprietăţi formale ale acestor depend.
□orice multime *D*□cu proprietatea ca orice dependenţă a mulţimii *D*este derivabilă din *D*□prin aplicarea proprietăţilor *p*1, *p*2, ..., *pr*, se numeste **acoperire** a lui *D*in raport cu proprietăţile *p*1, *p*2, ..., *pr*
(4)mulţimea *D*□se numeste **acoperire minimală**pentru *D*
□nu există nicio submulţime proprie, nevidă a lui *D*□care să fie o acoperire pentru *D*

Definitii (multime de axiome)
Fie *D* o multime de dependente functionale si D* inchiderea sa:
(1)O **mulţime de axiome** se numeste **completă**

□este suficienta pentru a obtine, plecand de la *D,* toate dependenţele din *D**
(2) O **mulţime de axiome** se numeste **închisă**
□orice dependenta dedusa pe baza ei, plecand de la *D* , face parte *D**.

Axiomele lui Armstrong
Fie *X, Y, Z, W*mulţimi de atribute ale unei entitati:
Ax. 1–reflexivitate:
$X \rightarrow X$
Ax. 2–creşterea determinantului:
dacă $X \rightarrow Y$şi *X inclus Z,* atunci $Z \rightarrow Y$
dacă $X \rightarrow Y$şi *W* inclus *Z,* atunci $X \cup Z \rightarrow Y \cup W$
dacă $X \rightarrow Y$atunci $X \cup Z \rightarrow Y \cup Z$
Ax. 3–tranzitivitate:
dacă $X \rightarrow Y$şi $Y \rightarrow Z$, atunci $X \rightarrow Z$.

Teorema (J.D. Ullman)
Axiomele Ax.1–Ax.3 reprezintă o mulţime închisă şi completă de axiome
Corolar
Închiderea unei multimi de dependente *D*consta din mulţimea dependenţelor deduse din *D* prin aplicarea axiomelor lui Armstrong.

Reguli de inferenta pentru dependente funct.
Fie *X, Y, Z, W*mulţimi de atribute ale unei entitati:
Reg. 1–descompunere:
dacă $X \rightarrow Y$ si *Z* inclus *Y*, atunci $X \rightarrow Z$
Reg. 2–reuniune:
dacă $X \rightarrow Y$ si $X \rightarrow Z$, atunci $X \rightarrow YZ$
Reg. 3–semitranzitivitatea:
dacă $X \rightarrow Y$ si $YZ \rightarrow W$, atunci $XZ \rightarrow W$

Definitii(dependente echivalente)

(1)Fie *F* o mulţime de dependenţe funcţionale totale
□**închiderea sa pseudo-tranzitivă***F*+=
reuniunea mulţimilor dependenţelor funcţionale totale care pot fi obţinute din *F* folosind axioma de pseudo-tranzitivitate (sau, echivalent, axiomele lui Armstrong si cele 3 reguli);
(2)Fie *F1*si *F2*două mulţimi de dependenţe funcţionale totale
□*F1*□*F2*ddacă *F1* + □*F2*+

(*F1*si *F2*sunt **echivalente**□inchiderile lor pseudotranzitive coincid);
(3)Fie *F* o mulţime de dependenţe funcţionale totale asociată unei mulţimi de atribute *A:*

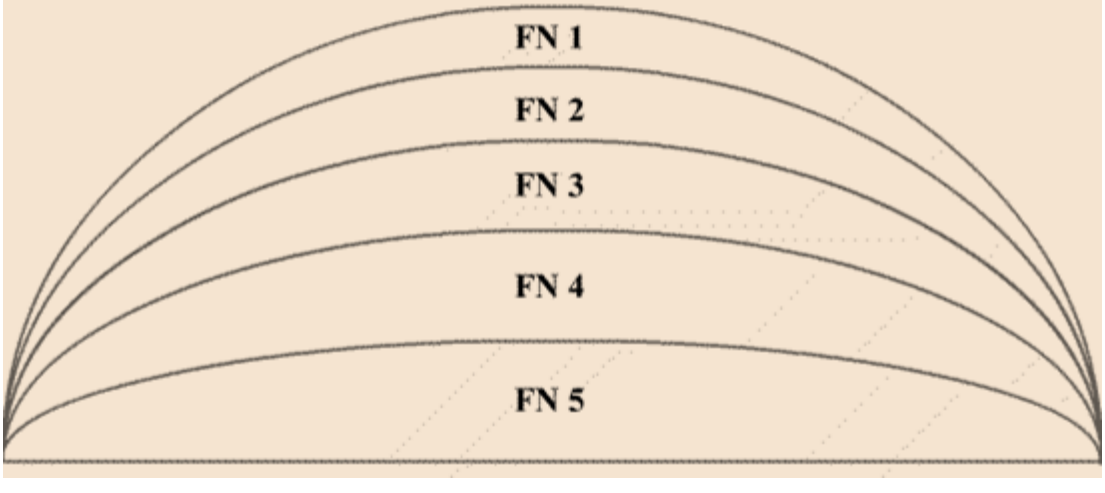
F defineşte o **acoperire minimală**dacă satisface următoarele proprietăţi:
•nicio dependenţă funcţională din *F*nu este redundantă,
•toate dependenţele funcţionale totale între submulţimi ale lui *A* se afla în închiderea pseudo-tranzitivă a lui *F;*

Fie *F*o multime de dependenţe funcţionale ale unei entitati:
□*F*este in **forma canonica** =
i.orice dependenta functionala din *F*are in membrul stang un singur atribut,
ii.F este minimala (nu contine dependente redundante);

20. Normalizarea relaţiilor

Cele 5 forme normale au un caracter progresiv:
ex.: o relaţie aflată în FN3 este automat în FN2 şi deci şi în FN1;
•Din punctul de vedere al modelului relaţional, singura formă normală obligatorie pentru toate relaţiile din BD este FN1;

dacă însă dorim să evităm toate anomaliile de actualizare (analizate mai sus) este necesar să continuăm procesul de normalizare cel puţin până la FN3;
•Din punct de vedere al performanţelor în exploatare, este preferabil ca BD să fie lăsată intr-o formă normală inferioară (se execută procesul invers normalizării: **denormalizare**a BD).



Normalizarea BD
•un proces de ameliorare progresivă a schemei conceptuale,
prin care un set de relaţii care încalcă anumite principii de proiectare este înlocuit cu un alt set de relaţii adecvat, coerent şi bine structurat;
•acest proces trebuie să satisfacă următoarele cerinţe:
- să garanteze **conservarea datelor**
- să garanteze **conservarea dependenţelor**dintre date,
- să reprezinte o **descompunere minimală**a relaţiilor iniţiale

Normalizarea BD(cont.)
•exista 2 metode de modelare a BD fara anomalii si fara pierdere de informatii:
(1)top-down
(2)bottom-up

Definitie formala (metoda *top-down*)
= procesul prin care relatia universala care modeleaza o situatie reala si **respecta restrictiile contextuale** incalcand astfel regulile de integritate este inlocuita cu un set de reguli din ce in ce mai adecvate, coerente si bine structurate;
•se realizează plecând de la o relaţie universală ce conţine toate atributele sistemului de modelat;
•se desfăşoară în mai mulţi paşi;
•fiecare pas (cu excepţia aducerii BD la FN1) presupune:
•identificarea dependenţelor funcţionale,
•verificarea îndeplinirii unor anumite proprietăţi denumite generic **forme normale**.

Orice formă normală se obţine aplicând **o schemă de descompunere**
•Există două tipuri de descompuneri:
I.Descompuneri care conservă dependenţele
relatia *R*este descompusa într-o mulţime de proiecţii *R*1, *R*2, ..., *Rk*a.î. dependenţele relatiei initiale *R*sunt echivalente (au închideri pseudo-tranzitive identice) cu reuniunea dependenţelor noilor relatii *R*1, *R*2, ..., *Rk*
II.Descompuneri fără pierderi de informaţie(*L-join*)
relatia *R*este descompusa într-o mulţime de proiecţii *R*1, *R*2, ..., *Rk*a.î. pentru orice realizare a lui *R*este adevărată relaţia:
R= JOIN(Π*B*1(*R*), Π*B*2(*R*), ...,Π*Bj*(*R*))
Regula Casey-Delobel
•descrie conditia ca o descompunere utilizată în procesul normalizării sa se efectueze fără pierdere de informaţie:

”dacă este satisfăcută o anumită dependenţă funcţională, atunci există o descompunere fără pierderi”;
•Fie: *R*(*A*) o schemă relaţională,
α, β, γ o partiţie a multimii de atribute *A*a.i.
α determină funcţional pe β iar γ contine restul atributelor in A;
atunci:
R(*A*) = JOIN(Πα□β(*R*), Πα□γ(*R*))
unde α□β reprezintă mulţimea atributelor care intervin în dependenţele funcţionale,
α□γ reprezintă reuniunea determinantului (atributul comun al compunerii) cu restul atributelor lui *A*.

O relaţie *R* este în *FN1* dacă fiecărui atribut care o compune îi corespunde o valoare indivizibilă (atomică).

În plus, o relatie nu trebuie să conţină atribute sau grupuri de atribute repetitive
Această formă figurează ca cerinţă minimală în majoritatea sistemelor relaţionale;
Algoritm AFN1

(aducerea unei relaţii în *FN1*prin eliminarea atributelor compuse şi a celor repetitive)
1.se introduc în relaţie, în locul atributelor compuse, componentele acestora,
2.se plasează grupurile de atribute repetitive, fiecare în câte o nouă relaţie,
3.se introduce în schema fiecărei noi relaţii de la pasul 2 cheia primară a relaţiei din care a fost extras atributul repetitiv,
4.se stabileşte cheia primară a fiecărei noi relaţii create la pasul 2,
aceasta este compusă din cheia introdusă la pasul 3, precum şi din atribute proprii ale acestor noi relaţii.

O relaţie *R* este în *FN2* ddacă:
•relaţia *R*este în *FN1*
•fiecare atribut care nu participă la cheia primară este dependent de întreaga cheie primară;
Se poate aplica regula Casey-Delobel:
fie relaţia *R*(*K*1, *K*2, *X*, *Y*);
unde *K*1 şi *K*2 definesc cheia primară
X şi *Y*sunt mulţimi de atribute astfel încât *K*1 *X*;
observam ca R nu este in FN2;
dar R poate fi înlocuita (fără pierdere de informaţie) cu doua proiecţii:
*R*1(*K*1, *K*2, *Y*) şi
*R*2(*K*1, *X*).

Algoritm AFN2
(aducerea unei relaţii în *FN2*prin eliminarea dependenţelor funcţionale parţiale din cadrul unor relaţii aflate în *FN1*)
1.pentru fiecare dependenţă funcţională parţială se creează o nouă relaţie având schema formată din determinantul şi determinatul acestei dependenţe
2.se elimină din cadrul relaţiei iniţiale atributele care formează determinatul dependenţei parţiale
3.dacă în relaţia iniţială există mai multe dependenţe parţiale cu acelaşi determinant, pentru acestea se creează o singură relaţie cu schema formată din determinant (luat o singură dată) şi din determinaţii dependenţelor considerate
4.se determină cheia primară a fiecărei noi relaţii create; aceasta va conţine atributele din determinantul dependenţei funcţionale parţiale care au stat la baza constituirii relaţiei
5.dacă noile relaţii create conţin dependenţe parţiale, atunci se face reia de la pasul 1, altfel STOP

O relaţie *R* este în *FN3* ddacă:
•relaţia *R*este în *FN2*
•fiecare atribut care nu participă la o cheia candidat este dependent direct de cheia primară;
A doua condiţie interzice utilizarea dependenţelor funcţionale tranzitive în cadrul relaţiei *R*
o relaţie este în FN3 dacă şi numai dacă fiecare atribut care nu este cheie depinde de cheie, de întreaga cheie şi numai de cheie.
Se poate aplica regula Casey-Delobel:

fie relaţia *R*(*K*, *X*1, *X*2, *X*3),
unde *K* este cheia primară a lui *R si*
atributul *X*2depinde tranzitiv de *K*,
presupunem că *K* *X*1 *X*2.

dependenţa funcţionala $X1 \rightarrow X2$care arată că *R*nu este în FN3
□se înlocuieşte *R*(fără pierdere de informaţie) prin două proiecţii

*R*1(*K*, *X*1, *X*3) şi *R*2(*X*1, *X*2).
□Dependenţa tranzitivă poate fi mai complexă:

fie *K*1o parte a cheii *K*,
tranzitivitatea poate fi de forma $K \rightarrow Y \rightarrow X2$unde $Y = \{K1, X1\}$
in acest caz, *R*poate fi descompusă în *R*1(*K*, *X*1, *X*3) şi *R*2(*K*1, *X*1, *X*2).

Algoritm AFN3
(aducerea unei relaţii *FN2*în *FN3*prin eliminarea dependenţelor funcţionale tranzitive)
1.pentru fiecare dependenţă funcţională tranzitivă se transferă atributele implicate în dependenţa tranzitivă într-o nouă relaţie
2.se determină cheia primară a fiecărei noi relaţii create la pasul 1
3.se introduc în relaţia iniţială, în locul atributelor transferate, cheile primare determinate la pasul 2
4.se reanalizează relaţia iniţială; dacă în cadrul ei există noi dependenţe tranzitive, atunci se reia de la pasul 1, altfel STOP

FN3 poate fi obţinută şi cu ajutorul unei **scheme de sinteză.**
Informal:
algoritmul de sinteză construieşte o acoperire minimală *F*a dependenţelor funcţionale totale:
•se elimină atributele şi dependenţele funcţionale redundante.
•mulţimea *F*este partiţionată în grupuri *Fi*, astfel încât:
•în fiecare grup *Fi*se află dependenţe funcţionale care au acelaşi membru stâng şi
•nu există două grupuri având acelaşi membru stâng
•fiecare grup *Fi*produce o schemă FN3
□Algoritmul realizează o descompunere ce conservă dependenţele.

Algoritm SNF3
(aducerea unei relaţii în *FN3*prin utilizarea unei scheme de sinteză)
1.se determină *F*, o acoperire minimală a lui *D*(mulţimea dependenţelor funcţionale)
2.se descompune mulţimea *F*în grupuri notate *Fi*, astfel încât în cadrul fiecărui grup să existe dependenţe funcţionale având aceeaşi parte stângă
3.se determină perechile de chei echivalente (*X*, *Y*) în raport cu *F*
4.pentru fiecare pereche de chei echivalente:
•se identifică grupurile *Fi*şi *Fj*care conţin dependenţele funcţionale cu partea stângă *X* şi respectiv *Y*
•se formează un nou grup de dependenţe *Fij*, care va conţine dependenţele funcţionale având membrul stâng (*X*, *Y*)
•se elimină grupurile *Fi*şi *Fj*, iar locul lor va fi luat de grupul *Fi j*
5.se determină o acoperire minimală a lui *F*, care va include toate dependenţele $X \rightarrow Y$,

unde *X*şi *Y*sunt chei echivalente (celelalte dependenţe sunt redundante)
6.se construiesc relaţii *FN3*(câte o relaţie pentru fiecare grup de dependenţe funcţionale).

	Condiţie de verificat	***Soluţie (normalizare)***
FN1	□Toate atributele relaţiei trebuie să fie atomice	□Fiecare atribut neatomic se transformă intr-o nouă relaţie □Se stabilesc relaţiile necesare între

		noile relaţii şi relaţia iniţială modificată
FN2	□Relaţia este în FN1; □Cheia sa primară constă din mai multe atribute; □Toate atributele care nu fac parte din cheia primară sunt complet dependente funcţional de cheia primară	□Fiecare parte a cheii primare, împreună cu atributele care depind funcţional complet de ea formează o nouă relaţie; □Se stabilesc relaţiile necesare între noile relaţii care au înlocuit-o pe cea iniţială
FN3	□Relaţia este în FN2; □Nici un atribut care nu face parte dintr-o cheie candidat nu este funcţional dependent de un alt atribut care nu face nici el parte dintr-o cheie candidat (nici un atribut care nu face parte dintr-o cheie candidat nu este funcţional dependent de cheia primară prin tranzitivitate)	□Se păstrează în relaţia iniţială numai cheia primară şi atributele care depind funcţional de ea direct (inclusiv atributul "incriminat"); □Se creează câte o nouă relaţie din fiecare atribut care nu face parte din cheia primară împreună cu toate atributele (care nu fac nici ele parte din cheia primară a relaţiei iniţiale) care sunt dependente funcţional de acesta; □Se stabilesc relaţiile necesare între noile relaţii şi relaţia iniţială modificată

Forma normala 4(*FN4*)

□Se bazeaza pe un alt tip de dependenta intre valorile atributelor din relatii: dependenta multivaloare;
□Acest tip de dependenta apare –in general –in relatiile in care:
•mai mult de un atribut prezinta valori multiple si
•intre acele atribute exista relatii de tip 1:m INDEPENDENTE, impuse de context.

Dependenta multivaloare = (***MVD= multi-valued dependency***)
= o dependenta intre minimum 3 atribute apartinand aceleiasi relatii *R*cu proprietatea ca:
(i) pentru fiecare valoare a lui *A*exista un set de valori ale lui *B*si un set de valori ale lui *C*
(ii) aceste seturi de valori sunt independente unul de celalalt.
□Notatie
(*A* →→*B, A* →→*C*)
□Exemplu
(nrG →→tipAutoReparat, nrG →→Client).
Definitii
O dependenta multivaloare *A*→→*B*dintr-o relatie *R*se numeste **triviala**□
(a)B inclus in *A***sau**

(b)A U *B* = *R;*

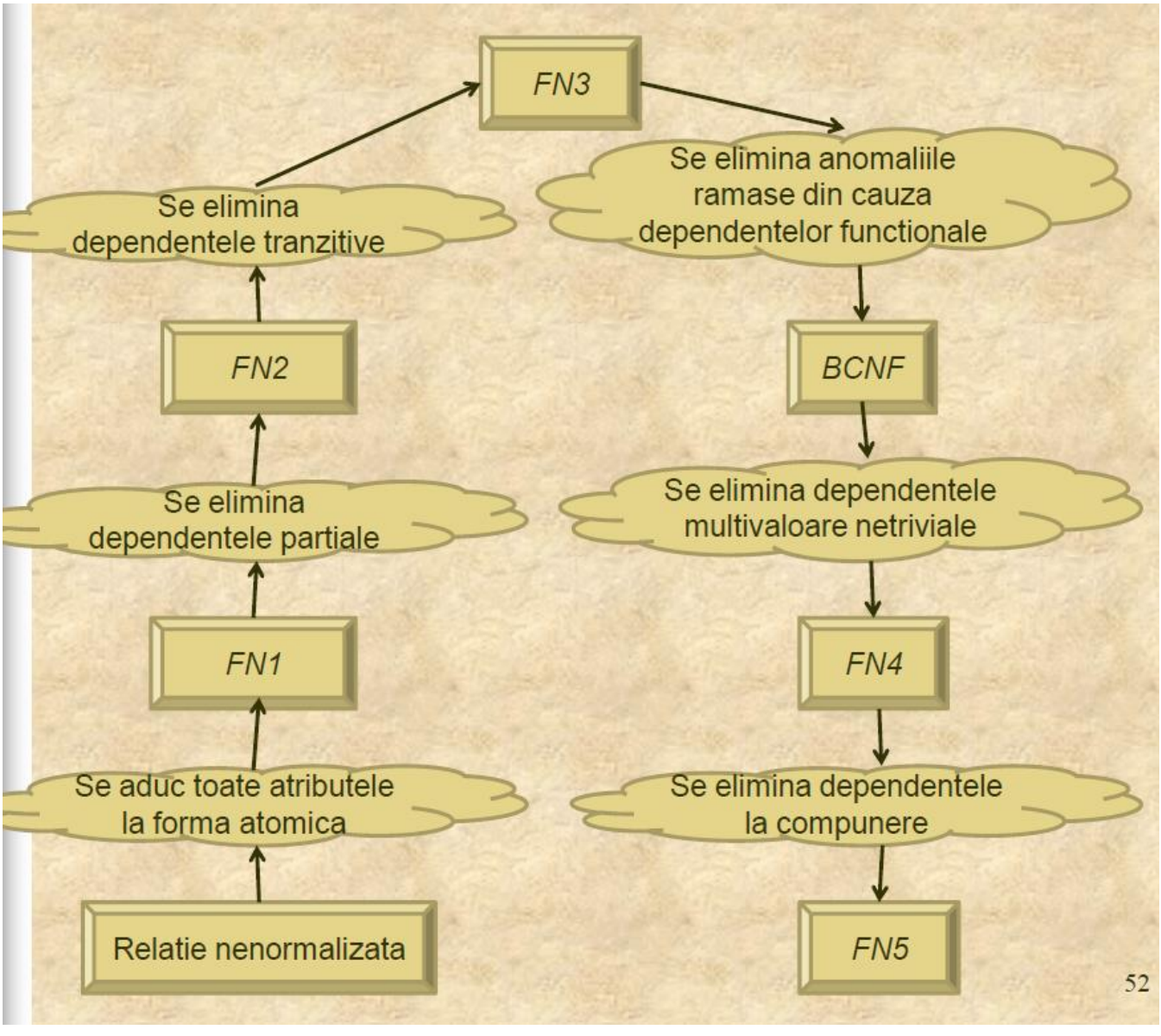
O dependenta multivaloare *A*→→*B*dintr-o relatie *R*se numeste **netriviala**
niciuna dintre cele 2 conditii de mai sus nu are loc;

Forma normala 5(*FN5*)
Se bazeaza pe un alt tip de dependenta intre valorile atributelor din relatii: dependenta la descompunerea fara pierdere de informatie (la descompunerea nonaditiva)
Acest tip de dependenta apare atunci cand o relatie trebuie descompusa in mai mult de 2 relatii (ca in cazul *FN4*) si este rezolvata prin *FN5*
Dependenta ladescompunerea fara pierdere de informatie (dependenta la descompunerea nonaditiva)= (***Lossless-join dependency***)

= o proprietate a operatiei de descompunere care impiedica aparitia de linii nelegitime atunci cand are loc o operatie de compunere naturala a mai multor relatii
Notatie

***(*R1, R2, … , Rk*)
O relatie *R*este in *FN5*ddaca
relatia *R*nu prezinta nicio dependenta la compunere

Concluzii
Normalizarea
= o tehnica de obtinere a unei multimi de relatii inzestrate cu anumite proprietati, in conformitate cu constrangerile specifice, impuse de situatia concreta modelata prin baza de date,
= o metoda formala prin care se pot identifica relatiile cu ajutorul campurilor de cheie şi se pot descoperi diversele tipuri de dependente care exista intre atributele lor,
= un proces care transforma o relatie trecand-o dintr-o *FN*in alta;
•la fiecare pas, se incearca eliminarea acelor caracteristici ale relatiei care o fac vulnerabila la anomaliile de actualizare
•trecerea intr-o *FN*superioara face a relatia mai invulnerabila dar şi mai restrictiva ca format:



Concluzii(cont.)**:**

FN1 →FN2

•elimină redundanţele datorate dependenţei netotale a atributelor care nu participă la o cheie, faţă de cheile lui *R*

•se suprimă dependenţele funcţionale care nu sunt totale;

FN2 →FN3

•elimină redundanţele datorate dependenţei tranzitive

•se suprimă dependenţele funcţionale tranzitive;

•se conserva si datele si dependentele;

FN3 →BCNF

•elimină redundanţele datorate dependenţei funcţionale

•se suprimă dependenţele în care partea stângă nu este o supercheie

•se conserva doar datele.

BCNF →*FN4*
•elimină redundanţele datorate multidependenţei
•se suprimă toate multidependenţele care nu sunt şi dependenţe funcţionale
•se conserva doar datele;

FN4 →*FN5*
•elimină redundanţele datorate dependenţei la compunere
•se suprimă toate dependenţele la compunere care nu sunt implicate de o cheie;

BCNF, FN4 şi*FN5*
•toate *FN* se bazeaza pe regula "orice determinant este o cheie",
•pentru fiecare *FN* determinantul se defineşte in raport cu un alt tip de dependenta:
□dependenţa funcţională, (*BNCF*)
□multidependenţa *(FN4)*
□dependenţa la compunere *(FN5).*

21. Denormalizarea relaţiilor

Deormalizarea
Fie R = {*R*1, *R*2, ..., *Rp*} o mulţime de relaţii
Denormalizarea *R* înseamnă înlocuirea *R*cu
R'= JOIN(*R*1, *R*2, ..., *Rp*),
astfel încât □1□i□p: proiecţia lui *R*după atributele lui *Ri*va produce din nou relaţia *Ri*
Observatii
Denormalizare □rafinarea schemei relaţionale a.î. gradul de normalizare a unei relaţii modificate să fie mai mic decât gradul de normalizare a cel puţin uneia dintre relaţiile iniţiale;
□Obiectivul denormalizarii
mărirea redundanţei (relaţia *R'*se află la un nivel de normalizare mai scăzut decât relaţiile *R*1, *R*2, ..., *Rp*compoente)
□reducerea numărului de *join*-uri care trebuie efectuate pentru rezolvarea unei interogări, prin realizarea unora dintre acestea în avans (ca parte din proiectarea bazei de date).

Problemele denormalizarii
1.Reaparitia anomaliilor pe care diversele forme normale reusisera sa le elimine;
2.Urmari negative ale denormalizarii asupra fisierelor stocate: un design fizic poate fi bun pentru anumite aplicaţii, dar prost pentru altele(in tabelul obtinut prin denormalizare -*join*–liniile par adiacente dar in memorie inregistrarile nu sunt! --> interogarile care vizeaza inregistrari numai in unul dintre tabelele participante la *join* vor fi mai lente);
3.Lipsa unui criteriu formal pt stabilirea nivelului la care denormalizarea trebuie sa se opreasca;
4.Lipsa unor reguli formale pentru stabilirea situaţiilor în care este indicată denormalizarea relaţiilor.

Cazuri in care denormalizarea este indicata:
Normalizarea "completa" a BD □scaderea semnificativa a performantelor BD
□3 exemple de situatii in care trebuie considerata posibilitatea reducerii gradului de normalizare in favoarea cresterii performantelor:
A.Actualizari vs. interogari

Dacă o relaţie are:
•o rată de reactualizare scăzută

•o rată de interogare foarte ridicată
atunci denormalizarea este o solutie.
B.Calcularea vs. memorarea datelor derivate
Din perspectiva proiectării fizice a BD se poate opta intre:
•stocarea in BD a atributelor derivate
•calcularea atributelor derivate de fiecare dată când este necesar

Criterii:
•costul memorarii vs costul recalcularii
•costul menţinereii concordanţei dintre datele calculate si datele operatioanle din care sunt derivate,
C.Pro sau contra redundantei

Din perspectiva proiectării fizice a BD se poate opta intre:
•eliminarea completa a redundantei (aducerea la FN5)
•dublarea unor atribute (pentru simplificarea *join*-urilor)

dupa evaluarea costurilor /pericolelor implicate de necesitatea actualizarii mai multor copii ale aceleasi informatii.

22. SQL (caracteristici, implementare, securitate, Oracle SQL)
SQL (*Structured Query Language*) este limbajul standard de tratarea sistemelor relaţionale
SQL limbaj neprocedural, declarativ
Utilizatorii specific CE trebuie obţinut, nu CUM
=> compilatorul limbajului *SQL* generează automat o procedură care accesează BD şi execută comanda
□*SQL*=limbaj relational (nivel de abstractizare mai ridicat□productivitate superioara)

Bazat pe
•algebra relationala,
•calculul relational,
•etc., in functie de caracteristicile necesare.

3 metode pt.implementarea limbajului SQL:
1.apelarea directă (*Direct Invocation*): constă în introducerea instrucţiunilor direct de la prompter (interactiv)
2.modularizarea (*Modul Language*): se folosesc proceduri apelate de programele aplicaţie
3.încapsularea (*Embedded SQL*): instrucţiunile *SQL* pot fi combinate cu instrucţiunile limbajului de programare al programului respectiv (Standardul *SQL*include suport pentru limbajele *C, C++, COBOL, Java, Ada, M, Fortran, Pascal, PL/1*)

SQL şi securitatea/ integritatea datelor
2 abordări ale problemei securităţii datelor, acceptatede oriceSGBD:
□controlul discreţionar
•accesul la un anumit obiect este la discreţia proprietarului acestuia,
□controlul obligatoriu
•fiecareobiect are un nivel de clasificare,
•fiecare utilizator are un nivel de permisiune

Oracle SQL
SQL= asigură comunicarea cu serverul*Oracle*
=>reduce timpul necesar creării şi întreţinerii aplicaţiilor de bazede date,
Oracle SQL =include extensii ale limbajului*SQL* standard *ANSI/ISO*
Instrumentele şi aplicaţiile *Oracle* =furnizează instrucţiuni suplimentare.
Utilitarele permit
•executarea instrucţiunilor limbajului *SQL* standard asupra unei baze de date *Oracle,*
•executarea instrucţiunilor saufuncţiilor suplimentare disponibile.

23. Clasificarea comenzilor SQL (LDD, etc.)

Instrucþiunile SQLse împart în mai multe categorii, înfuncþie de tipul acþiunii pe care o realizează:
a.limbajul de definirea datelor(LDD);
b.limbajul de prelucrarea datelor(LMD);
c.limbajul de control al tranzacþiilor(LCT);
d.limbajul de control al datelor(LCD);
e.comenzispeciale:
  i.instrucþiunile pentru controlul sesiunii,
  ii.instructiuni pentru controlul sistemului,
  iii.instrucþiunile SQL încapsulate.

(a)Limbajul de definirea datelor(LDD):(LDD contine comenzi pt. definirea structurii obiectelor unei scheme):
•specific fiecărui SGBD
•funcþiile principale sunt aceleaºi
•conceptual, LDD realizează:
•definirea entitãþilor ºi a atributelor acestora prin: nume, formã de memorare, lungime
•precizarea relaþiilor dintre date ºi strategiile de acces la ele
•stabilirea criteriilor diferenþiate de confidenþialitate
•stabilirea criteriilor de validare automatã a datelor utilizate.

(b)Limbajul de prelucrarea datelor(LMD):(LMD contine comenzipt. interogarea si prelucrarea datelor din obiectele unei scheme);
•permite formalizarea operaþiilor care trebuie executate asupra unei baze de date sub forma unor comenzi
•o comandă are următoarea structură:
•operaþia (deschidere/închidere, calcul aritmetic/logic, editare, extragere, adãugare, ºtergere, cãutare, reactualizare etc.),
•criterii de selecþie,
•mod de acces (secvenþial, indexat etc.),
•format de editare;
•2 tipuri de LMD:
•procedurale (specificã modul în care se obþine rezultatul unei comenzi LMD)
•neprocedurale(descriu doar datele ce vor fi obþinute ºi nu modalitatea de obþinere a acestora).

(c) Limbajulde control al tranzactiilor(LCT):
contine comenzipt. gestionarea modificãrilo refectuate de cãtre comenzile LMD ºi grupează aceste comenzi în unitãþi logice, numite tranzacþii:
?salvarea modificãrilor unei tranzacþii (COMMIT);

?anularea modificãrilor dintr-o tranzacþie fie în întregime, fie începând de la un punct intermediar (ROLLBACK);
?definirea unui punct intermediar pânã la care tranzacþia poate fi anulatã(SAVEPOINT);
?stabilirea de proprietãþi ale tranzacþiei (SET TRANSACTION TO).

(d) Limbajulde control al datelor(LCD):
contine comenzi pt. gestionarea accesului la date (asigurarea confidenþialitãþii ºi integritãþii datelor, salvarea informaþiei în cazul unor defecþiuni, obþinerea unor performanþe, rezolvarea unor probleme de concurenþã):
?acordarea de privilegii ºi role-uri (GRANT);
?revocarea privilegiilor ºi role-urilor acordate (REVOKE).

(e) Comenzispeciale:
i.Instrucþiunile pentru controlul sesiunii
ii.Instrucþiunile pentru controlul sistemului
iii.Instrucþiunile SQL încapsulate.

i.Instrucþiunile pentru controlul sesiunii
permit gestionarea proprietãþilor sesiunii unui utilizator; ex.
?ALTER SESSION:permite modificarea sesiunii curente, astfel încât aceasta sã îndeplineascã funcþiuni specializate,
?SET ROLE:determinã activarea sau dezactivarea role-urilor pentru sesiunea curentã;

ii.Instrucþiunile pentru controlul sistemului
?ALTER SYSTEM permite controlul sistemului, modificand, în mod dinamic, proprietãþile instanþei serverului Oracle (schimbarea anumitor setãri: numãrul minim de servere partajate,restricþionarea sau suprimarea unei sesiuni, golirea zonei shared pooldin SGA, suspendarea tuturor operaþiilor I/Oetc.).

iii.InstrucþiunileSQLîncapsulate
= sunt reprezentare de comenzi LDD, LMD ºi LCT care pot fi încorporate înprograme scrise în limbaje procedurale, urmând sã fie utilizate prin intermediul precompilatoarelor Oracle
permit:
?definirea, alocarea ºi eliberarea cursoarelor (DECLARE CURSOR, OPEN, CLOSE),
?specificarea unei bazede date ºi conectarea la sistemul Oracle (DECLARE DATABASE, CONNECT),
?declararea de variabile (DECLARE STATEMENT),
?iniþializarea de descriptori (DESCRIBE),
?specificarea modului încare urmeazã sã fie tratate erorile ºi avertismentele (WHENEVER),
?analizarea ºi executarea instrucþiunilor SQL (PREPARE, EXECUTE, EXECUTE IMMEDIATE),
?regãsirea informaþiilor din baza de date (FETCH).

24. Procesarea comenzilor ºi interogarilor SQL
Instrucþiunile SQLse împart în mai multe categorii, înfuncþie de tipul acþiunii pe care o realizeazã:
a.limbajul de definirea datelor(LDD);
b.limbajul de prelucrarea datelor(LMD);
c.limbajul de control al tranzacþiilor(LCT);
d.limbajul de control al datelor(LCD);
e.comenzispeciale:
  i.instrucþiunile pentru controlul sesiunii,
  ii.instructiuni pentru controlul sistemului,

iii.instrucþiunile SQL încapsulate.

(a)Limbajul de definirea datelor(LDD):(LDD contine comenzi pt. definirea structurii obiectelor unei scheme):
•specific fiecãrui SGBD
•funcþiile principale sunt aceleaºi
•conceptual, LDD realizeazã:
•definirea entitãþilor ºi a atributelor acestora prin: nume, formã de memorare, lungime
•precizarea relaþiilor dintre date ºi strategiile de acces la ele
•stabilirea criteriilor diferenþiate de confidenþialitate
•stabilirea criteriilor de validare automatã a datelor utilizate.

(b)Limbajul de prelucrarea datelor(LMD):(LMD contine comenzipt. interogarea si prelucrarea datelor din obiectele unei scheme);
•permite formalizarea operaþiilor care trebuie executate asupra unei baze de date sub forma unor comenzi
•o comandã are urmãtoarea structurã:
•operaþia (deschidere/închidere, calcul aritmetic/logic, editare, extragere, adãugare, ºtergere, cãutare, reactualizare etc.),
•criterii de selecþie,
•mod de acces (secvenþial, indexat etc.),
•format de editare;
•2 tipuri de LMD:
•procedurale (specificã modul în care se obþine rezultatul unei comenzi LMD)
•neprocedurale(descriu doar datele ce vor fi obþinute ºi nu modalitatea de obþinere a acestora).

(c) Limbajulde control al tranzactiilor(LCT):
contine comenzipt. gestionarea modificãrilo refectuate de cãtre comenzile LMD ºi grupeazã aceste comenzi în unitãþi logice, numite tranzacþii:
?salvarea modificãrilor unei tranzacþii (COMMIT);
?anularea modificãrilor dintr-o tranzacþie fie în întregime, fie începând de la un punct intermediar (ROLLBACK);
?definirea unui punct intermediar pânã la care tranzacþia poate fi anulatã(SAVEPOINT);
?stabilirea de proprietãþi ale tranzacþiei (SET TRANSACTION TO).

(d) Limbajulde control al datelor(LCD):
contine comenzi pt. gestionarea accesului la date (asigurarea confidenþialitãþii ºi integritãþii datelor, salvarea informaþiei în cazul unor defecþiuni, obþinerea unor performanþe, rezolvarea unor probleme de concurenþã):
?acordarea de privilegii ºi role-uri (GRANT);
?revocarea privilegiilor ºi role-urilor acordate (REVOKE).

(e) Comenzispeciale:
i.Instrucþiunile pentru controlul sesiunii
ii.Instrucþiunile pentru controlul sistemului
iii.Instrucþiunile SQL încapsulate.

i.Instrucþiunile pentru controlul sesiunii
permit gestionarea proprietãþilor sesiunii unui utilizator; ex.

?ALTER SESSION:permite modificarea sesiunii curente, astfel încât aceasta sã îndeplineascã funcþiuni specializate,
?SET ROLE:determinã activarea sau dezactivarea role-urilor pentru sesiunea curentã;

ii.Instrucþiunile pentru controlul sistemului
?ALTER SYSTEM permite controlul sistemului, modificand, în mod dinamic, proprietãþile instanþei serverului Oracle (schimbarea anumitor setãri: numãrul minim de servere partajate,restricþionarea sau suprimarea unei sesiuni, golirea zonei shared pooldin SGA, suspendarea tuturor operaþiilor I/Oetc.).

iii.InstrucþiunileSQLîncapsulate
= sunt reprezentare de comenzi LDD, LMD ºi LCT care pot fi încorporate înprograme scrise în limbaje procedurale, urmând sã fie utilizate prin intermediul precompilatoarelor Oracle
permit:
?definirea, alocarea ºi eliberarea cursoarelor (DECLARE CURSOR, OPEN, CLOSE),
?specificarea unei bazede date ºi conectarea la sistemul Oracle (DECLARE DATABASE, CONNECT),
?declararea de variabile (DECLARE STATEMENT),
?iniþializarea de descriptori (DESCRIBE),
?specificarea modului încare urmeazã sã fie tratate erorile ºi avertismentele (WHENEVER),
?analizarea ºi executarea instrucþiunilor SQL (PREPARE, EXECUTE, EXECUTE IMMEDIATE),
?regãsirea informaþiilor din baza de date (FETCH).

25. Optimizarea comenzilor SQL

Optimizarea comenzilor SQL
?constituie o etapa importantã în procesarea oricãrei instrucþiuni LMD
?existã mai multe posibilitãþi de a executa o astfel de instrucþiune(datorate, de exemplu, ordiniiîncare sunt accesate tabelele sau indecºii)
?se realizeaza
i.cu ajutorul unui modul software al sistemului Oracle numit optimizor,
ii.prin directivele (hint) date de catre proiectantul aplicatiei sub forma de comentarii atasate instructiunilor SQL.

Sistemul Oracle dispune de:
?optimizori pe bazã de cost (utilizati preponderent în versiunile cele mai recente ale lui Oracle Server)
?optimizori pe bazã de reguli (nu au fost actualizati în noile versiuni dar au ramas disponibil pentru compatibilitate).
ii Proiectantul unei aplicaþii deþine mai multe informaþii despre datele specifice acesteia=>
?el poate stabili cãi mai eficiente pentru execuþia instrucþiunilor SQL=>
?el include în textul comenzii SQL respective, sub formã de comentarii, niste directive (hint) de executie.

26. Obiectele unei BD Oracle (definitie, reguli de denumire)
Obiectele bazei de date
1.1. Generalitati
•O BD Oracle = { scheme }.
•O schemã = {obiecte}
•Obiect = o structura logica de date;

Oracle Database recunoaste 2 tipuri de obiecte:
a)obiecte care sunt asociate unei scheme particulare;
b)obiecte care nu aparþin nici unei scheme.

Reguli de denumire a obiectelor bazei de date Intr-o schema a Oracle Database trebuie denumite:
•toate obiectele,
•anumite parti componente ale anumitor obiecte:
?coloanele tabelelor sau vizualizãrilor,
?partiþiile ºi subpartiþiile tabelelor ºi indecºilor,
?constrângerile de integritate asupra tabelelor;

(R1)
• Numele obiectelor sunt unice la nivelul unei BD
• Într-o instrucþiune SQL, numele unui obiect poate fi reprezentat printr-un identificator încadrat, sau nu, între ghilimele
• Identificatorii nu pot fi cuvinte rezervate ale serverului Oracle;

(R2)
• Identificatorii trebuie sã înceapã cu o literã ºi sã aibã maximum 30 de caractere; Excepþii:
• numele BD (max. 8 caractere),
• numele legãturii unei BD (max. 128 caract.); (R3)
• Identificatorii pot conþine caractere alfanumerice ºi simbolurile „_”, „$”, „#”
• Ei trebuie sã înceapã cu un caracter alfabetic

(R4)
•Fiecare schemã din BD are propriul sãu spaþiu de nume (namespace) pentru obiectele pe care le conþine
•Douã obiecte din acelaºi spaþiu de nume al serverului Oracle nu pot avea acelaºi identificator.
?un tabel ºi o vizualizare din cadrul aceleiaºi scheme NU pot avea acelaºi nume, însã
?un tabel poate avea acelaºi nume cu un index.
Identificatorii obiectelor sunt case-sensitive doar în cazul în care sunt încadraþi între ghilimele;
• altfel, ei sunt convertiþi automat în majuscule.

27. Tipuri de date ºi literali SQL (definitii, clasificare, exemple, modele de format)
Tipuri de date SQL
2.1. Tipurile de date predefinite din sistemul Oracle:
a)caracter,
b)numeric,
c)datã calendaristicã ºi timp,
d) LOB (large objects) ºi
e)adrese unice ale liniilor din tabele.

a) Tipurile de date predefinite pentru stocarea ºirurilor de caractere sunt :
VARCHAR2 (n [BYTE |CHAR])
CHAR[(n [BYTE | CHAR])]
NVARCHAR2(n)
NCHAR[(n)]
iar pentru stocarea ºirurilor de caractere de dimensiuni mari sunt :
LONG,

LONG RAW
RAW.

b) Tipurile de date predefinite pentru stocarea valorilor numerice sunt :
NUMBER,
BINARY_FLOAT
BINARY_DOUBLE

c) Tipurile de date predefinite pentru stocarea datelor calendaristice ºi a momentelor de timp sunt :
DATE,
TIMESTAMP
INTERVAL

d)Tipurile de date predefinite pentru stocarea obiectelor mari LOB (large object) sunt :
CLOB
NCLOB
BLOB
BFILE

e) Tipurile de date predefinite pentru reprezentarea adreselor liniilor în tabele sunt :
ROWID
UROWID

2.Tipuri de date SQL
2.2. Tipuri de date ANSI, DB2 ºi SQL/DS acceptate în sistemul Oracle
Sistemul Oracle recunoaºte numele tipurilor de date specifice ANSI sau IBM, le converteste cf. unor conventii prestabilite si le utilizeaza pentru crearea tabelelor ºi clusterelor.

Tipurile de date definite de utilizator
Utilizatorul poate defini noi tipuri de date pe baza:
•oricaruia dintre tipurile de date predefinite,
•altor tipuri de date definite de utilizator,
•se obtin urmatoarele tipuri de date obiect care modeleaza structura ºi comportamentul datelor in aplicatii:
A.tipurile obiect,abstractizări ale entitãþilor din lumea realã, necesare în programele de aplicaþie,
•un tip obiect = o schemã obiect cu trei tipuri de componente:
•nume,
•atribute,
•metode.
B.tipurile referinþã (REF), conþin adrese logice ale liniilor obiect,
C.vectorii (varying array sau varray), modeleaza o mulþime ordonatã de elemente având acelaºi tip;
•fiecare element are un index, care reprezintã numărul corespunzător poziþiei elementului în vector;
D.tablourile imbricate (nested table) = modeleazã o mulþime neordonatã de elemente al caror tip de date poate fi:
•un tip predefinit,
•un tip definit de utilizator;

Literali
3.1. Literali de tip caracter

3.2. Literali numerici
3.3. Literali pentru date calendaristice
3.4. Literali marci de timp
3.5. Literali de intervale de timp;
?Definitie: Literal =
o valoare constanta pe perioda de viata a BD.

3.1. Literali de tip caracter
sintaxa:
se includ între apostrofuri (pentru a permite sistemului Oracle sã le distingã de numele obiectelor schemei);
•sunt considerate de tip CHAR (în expresii ºi condiþii);
•lungimea maximã admis: 4000 de octeþi.

3.2. Literali numerici
•de tip Number:
? in notatie clasica: precedate de + sau –
max . 38 caractere
? in notatie stiintifica: cu mantisa ºi exponent;

3.3. Literali pentru date calendaristice
•sunt formate din campuri
•formatul implicit:
•specificat prin param. de iniþializare NLS_DATE_FORMAT
•in notaþia ANSI:
•cuvantul-cheie DATE si
•formatul 'YYYY-MM-DD'

3.4. Literali pentru marci de timp
•sunt formate din campuri
•precizeaza anul, luna, ziua, ora, minutul, secunda ºi fracþiunile de secundã.
•in notaþia ANSI:
•cuvantul-cheie TIMESTAMP'
•formatul 'YYYY-MM-DD HH:MM:SS.FFF'

3.5. Literali pentru intervale de timp
•sunt formate din campuri; daca precizia compurilor este mai mare decat 2, trebuie specificata
•utilizate impreuna cu functiile analitice
•specifica o perioada de timp care poate fi exprimatã în
•ani ºi luni sau
•zile, ore, minute ºi secunde.
•2 tipuri de literale pentru specificarea intervalelor:
•YEAR TO MONTH ºi
•DAY TO SECOND

28. Comentarii SQL (pt utilizarea hinturilor în optimizari)
Optimizarea comenzilor SQL
?constituie o etapa importantã în procesarea oricãrei instrucþiuni LMD

?există mai multe posibilitãþi de a executa o astfel de instrucþiune(datorate, de exemplu, ordiniiîncare sunt accesate tabelele sau indecºii)
?se realizeaza
i.cu ajutorul unui modul software al sistemului Oracle numit optimizor,
ii.prin directivele (hint) date de catre proiectantul aplicatiei sub forma de comentarii atasate instructiunilor SQL.

Sistemul Oracle dispune de:
?optimizori pe bazã de cost (utilizati preponderent în versiunile cele mai recente ale lui Oracle Server)
?optimizori pe bazã de reguli (nu au fost actualizati în noile versiuni dar au ramas disponibil pentru compatibilitate).
ii Proiectantul unei aplicaþii deþine mai multe informaþii despre datele specifice acesteia=>
?el poate stabili cãi mai eficiente pentru execuþia instrucþiunilor SQL=>
?el include în textul comenzii SQL respective, sub formã de comentarii, niste directive (hint) de executie.

29. Pseudocoloane, operatori ºi functii SQL
Pseudocoloane
? Definitie: Pseudocoloana =
= se comportã ca ºi o coloanã a unui tabel, dar nu este stocatã efectiv într-un tabel
•Se pot face interogãri asupra pseudocoloanelor, dar valorile acestora nu se pot insera, actualiza sau ºterge

Exemple
ROWID
= returneazã adresa unei linii din baza de date, furnizând modul cel mai rapid de a accesa linia respectivã
ROWNUM
= returneazã numãrul de ordine al liniilor rezultate în urma execuþiei unei cereri
• poate fi utilizatã pentru a limita numãrul de linii returnate.

Definitie: Operatori SQL=
= prelucreazã date individuale (= operanzi = argumente);
•sunt reprezentaþi prin:
?caractere speciale
?cuvinte cheie
Clasificare
•unari (acþioneazã asupra unui singur operand)
•binari (acþioneazã asupra a doi operanzi).

Definitie: Expresie SQL =
= o combinaþie de una sau mai multe valori, operatori ºi funcþii SQL

? Definitie: Conditie SQL =
= combinã una sau mai multe expresii ºi operatori logici, returnând una dintre valorile TRUE, FALSE sau NULL.

Alti operatori ºi expresii SQL:
1.Expresii care folosesc valori de tip datã calendaristicã ºi interval:

operatorii utilizati ºi tipul rezultatelor obþinute sunt :
•Data + Interval
•Data – Interval
•Interval + Data
=> rezultatul este de tip datã calendaristicã,
•Data – Data
•Interval + Interval
•Interval – Interval
•Interval * Number
•Number * Interval
•Interval / Number
=> rezultatul este de tip interval.

2.Expresii de tip tablou imbricat ºi vector:
se pot aplica operatorii:
•egalitate,
•inegalitate,
•MULTISET EXCEPT => diferenta a 2 tablouri imbricate,
•MULTISET INTERSECT => intersectia a 2 tablouri imbricate,
•MULTISET UNION => reuniunea a 2 tablouri imbricate;
2.Obiecte LOB:
nu sunt acceptate în condiþiile de comparaþie (se pot insa folosi blocuri PL/SQL).

Funcþii SQL
1.Functii numerice;
2.Functii de tip caracter care returneaza valori de tip caracter;
3.Functii de tip caracter care returneaza valori numerice;
4.Functii de tip NLS (Natural language Support);
5.Functii de tip data calendaristica;
6.Functii generale de comparatie;
7.Functii de conversie intre tipurile de date SQL;
8.Functii XML;
9.Functii pentru prelucrarea valorilor Null;
10.Functii speciale.

Funcþiile SQL
•similare operatorilor (prelucreaza date, returneaza un rezultat);
•difera de operatori (pot avea 0, 1, oricate argumente);
•sunt predefinite în sistemul Oracle ºi pot fi utilizate în instrucþiuni SQL
•diferite de funcþiile definite de utilizator (scrise în PL/SQL);
•dacã o funcþie SQL este apelatã cu un argument având un alt tip de date decât cel predefinit, sistemul converteºte implicit argumentul înainte sã evalueze funcþia
•dacã o funcþie SQL este apelatã cu un argument null, ea returneazã automat valoarea null;
•exceptii: functiile
CONCAT
NVL REPLACE
REGEXP_REPLACE

Categoriile de funcþiile SQL :

a)funcþii single row;
b)funcþii agregat;
c)funcþii analitice;
d)funcþii referitoare la obiecte;
e)funcþii pe modele;
f)funcþii definite de utilizator.

a)Funcþiile single row:
•returnează o singură linie rezultat pentru fiecare linie a tabelului sau vizualizării interogate

b,c) Funcþiile agregat ºi analitice:
•necesare pt scrierea interogărilor;
d) Funcþiile referitoare la obiecte:
•funcþii care operează asupra valorilor de tip LOB:

e)Funcþii pe modele:
•functii care operează asupra modelelor construite cu ajutorul pachetului DBMS_DATA_MINING sau cu Oracle

f)Funcþiile definite de utilizator:
•pot fi create cu ajutorul limbajului PL/SQL
•sunt stocate în baza de date.

30. Limbaje pentru prelucrarea datelor relaţionale
Modelul relaþional incorporeaza mai multe limbaje de interogare (neprocedurale);
Clasificarea limbajelor de prelucrare a datelor relaþionale:
?limbaje algebrice–bazate pe teoria mulþimilor (SEQUEL, SQL);
?limbaje predicative–bazate pe calculul predicatelor:
  ?orientate pe tupluri (QUEL, ALPHA);
  ?orientate pe domenii (variabilele iau valori în domeniile relaþiilor, ºi nu în tuplurile acestora)
    ?non-grafice
    ?grafice
      ?cu variabile domeniu explicite (QBE);
      ?fără variabile domeniu explicite (LAGRIF, CUPID, VGQF).

Limbajele grafice oferă utilizatorului o imagine sau o ilustrare a structurii relaþiei
utilizatorul completează un exemplu cu ceea ce doreºte
sistemul returnează datele cerute în acest format.

SQL(Structured Query Language)
= limbajul standardde descriere a datelor ºi accesare a informaþiilor din BD
•creat ºi dezvoltat iniþial de către compania IBM Research,
•implementat în cadrul prototipului System R;
•exista peste o sută de dialecte

au fost create două tehnologii puternice:
A.API
B.ODBC.

Tehnologia API(Application Programming Interface) =
= o bibliotecã de funcþii SQLce se pot integra într-un program gazdã
•avantaj: este o tehnica de lucru familiara programatorilor
•dezavantaj: lipsa de interoperabilitate:

Tehnologia ODBC (Open Database Connectivity)
= o mulþime de primitive ce pune la dispoziþia utilizatorilor o interfaþã comunã pentru accesarea bazelor de date SQLeterogene
•creata in 1992, de Microsoft, pe baza limbajuluiC
•a devenit standardul industrial de facto
•avantaje:
(1) grad înalt de interoperabilitate:
(2) grad inalt de flexibilitate
instrucþiunile sql pot fi incluse explicit în codul sursã sau construite dinamic în timpul execuþiei.

Arhitectura interfetei ODBC include patru componente:
1.aplicaþia:
•efectueazã prelucrarea ºi apelurile la funcþiile ODBCpentru a transmite comenzile SQLcãtre SGBD ºi a regãsi rezultatele furnizate de cãtre acesta
2.administratorul de drivere:
•încarcã driverele în contul unei aplicaþii
3.agentul pentru drivere ºi baza de date:
4.sursa de date:
•este formatã din datele pe care doresc sã le acceseze:
? utilizatorul ºi SGBD-ul asociat
? SO gazdã ºi platforma de reþea (dacã existã).

Limbajele algebrice
= limbaje neprocedurale care utilizeazã relaþii pentru a transforma datele de intrare în datele de ieºiri solicitate

Exemple:
•SQUARE,
•SEQUEL
•SQL;

SEQUEL(Structured English as a Query Language) =
= limbaj algebric definit în 1974 de D. D. Chamberlin ºi R.F. Boyce, tot de la Laboratorul de cercetãri IBM, pentru prototipul relaþional System R.
•este singurul limbaj relaþional care are integratã închiderea tranzitivã
•operaþia fundamentalã a limbajului: SELECT
•o extensie comercialã a acestui limbaj: SQL (SEQUEL = parintele SQL).
10
1.Generalitati
2.Limbaje algebrice
3.Limbaje predicative
4.Utilizarea limbajelor de prelucrare a datelor relationale in contextul limbajelor de programare
5.SQL
6.Limbajul MD

Limbajele predicative
= limbaje bazate pe calculul cu predicate
Exemple:

QUEL= limbaj predicativ orientat pe tupluri
poate fi utilizat independent sau inclus în limbajul C.
Limbajul este caracterizat de:
•declararea unei variabile tuplu pentru fiecare relaþie (prin RANGE)
•absenþa cuantificatorilor în expresii;
•?: este reprezentatimplicit prin declaraþia RANGE, iar cuantificatorul universal este simulat cu ajutorul.
•?: este simulat cu ajutorul funcþiei COUNT
•utilizarea unor operatori speciali pe mulþimi.

QBE(Query By Example)
= un limbaj predicativ orientat spre domenii
•prezentat de M.M. Zloof în 1977 ºi comercializat de IBMdupã 1980, pentru a-i ajuta utilizatorii neiniþiaþi
•extrem de accesibil => limbaj furnizat (în diferite forme) de majoritatea SGBD-urilor, inclusiv Microsoft Access.
•reprezintã un mod de tratare virtual pentru accesarea informaþiilor dintr-o BD, prin utilizarea ºabloanelor de interogare
•limbajul dispune de primitive de programare graficã a cererilor de date:
?utilizatorii completeazã o mulþime de câmpuri predefinite intr-un ecran special.
?SGBD-ul construieºte în fundal instrucþiunea SQL.
•nu existã un standard oficial pentru aceastã clasã de limbaje
•funcþionalitatea este, în general, foarte asemãnãtoare
•limbajele sunt mai intuitive decât SQL

SQL(Structured Query Language) =
= limbaj neprocedural pentru interogarea ºi prelucrarea informaþiilor din BD
= limbaj declarativ:
•utilizatorul trebuie doar sã descrie CE si NU CUM trebuie obþinut
•compilatorul limbajului SQLgenereazã automat o procedurã care accesează BD ºi executã comanda
?SQLpermite:
•definirea, prelucrarea ºi interogarea datelor,
•controlul accesului la date
•la nivel logic si
•la nivel de multime
?Comenzile SQLpot fi integrate în programe scrise în alte limbaje, de exemplu Cobol, C, C++, Javaetc.

Clasificarea instrucþiunile SQL:
•limbajul de definire a datelor (LDD);
•limbajul de prelucrare a datelor (LMD);
•limbajul de control al datelor (LCD).
  •instrucþiuni pentru controlul sesiunii;
  •instrucþiuni pentru controlul sistemului;
  •instrucþiuni SQLîncapsulate

MDX(Multidimensional Expressions) =
= limbaj multidimensional specializat

Interogarile MDX=
= interogari care se bazeaza pe cuburi OLAP
•intorc ca rezultate date agregate, aflate la intersecþia oricãror niveluri de agregare din oricare ierarhii, din oricare dimensiuni;
?Cubul(UDM–Unified Dimensional Model) =
= o structurã specialã, logica nu fizica si care va fi interogatã în cadrul unui sistem multidimensional
•poate rãspunde rapid la interogãri care cer o anumitã agregare a datelor aflate la intersecþia oricãror niveluri din oricare ierarhii din oricate dimensiuni;
?Scopulunei interogãri MDX:
obþinerea unei mulþimi de date, numitã mulþime de celule(cellset) sau cub-rezultat (i.e.: o submulþime a cubului original).