Dezvoltarea sistemelor de baze de date

Dezvoltarea bazelor de date

- Colectarea si analiza cerintelor
- Proiectarea bazelor de date
  - Proiectarea conceptuala a bazelor de date
  - Alegerea unui model de date și a unui SGBD
  - Proiectarea logica a bazelor de date relaționale
  - Proiectarea fizica a bazelor de date relaționale
- Implementarea si testarea bazelor de date relaționale

Dezvoltarea aplicațiilor de baze de date

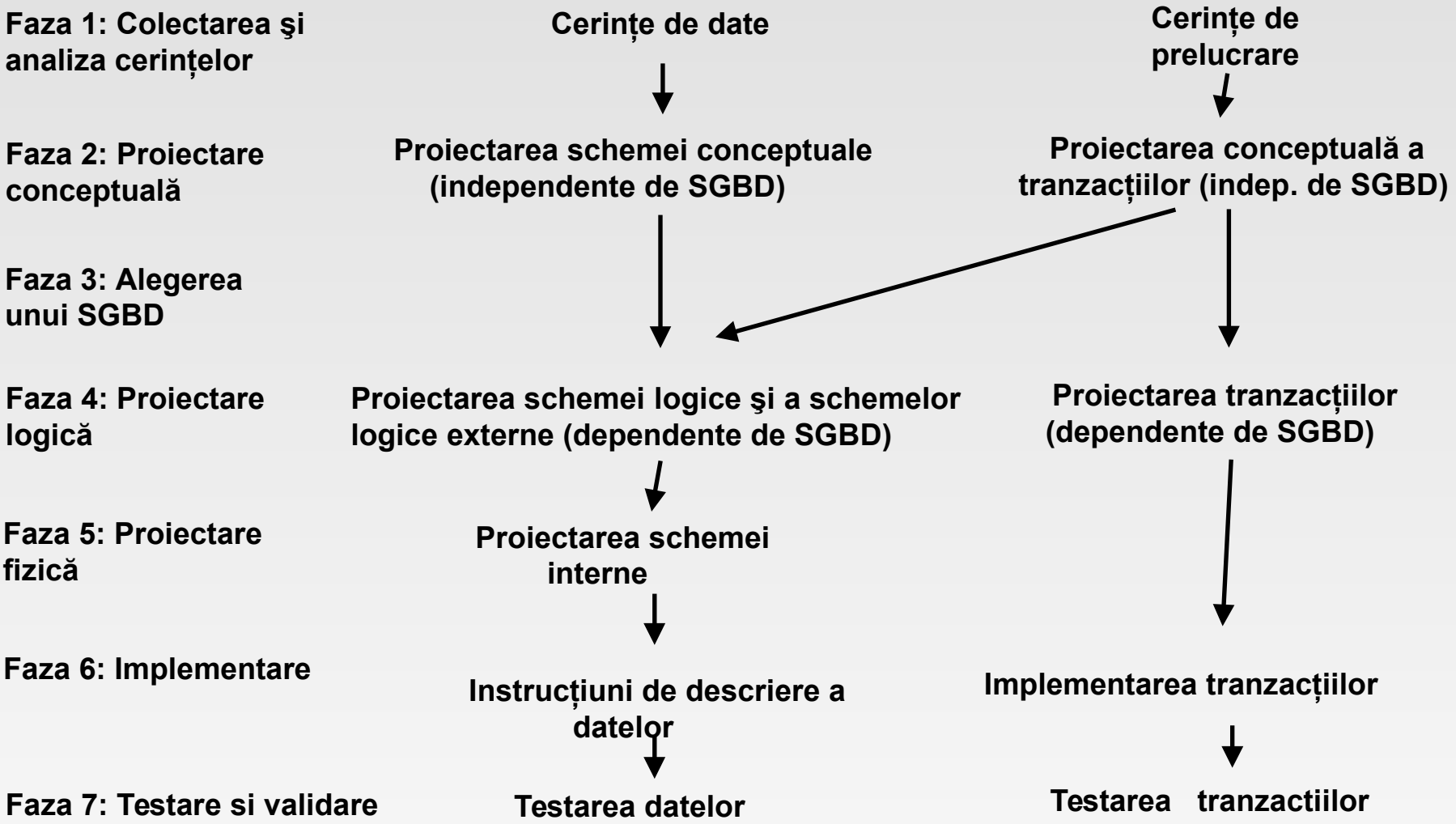
- Limbaje procedurale de extensie a limbajului SQL
  - Limbajele Transact-SQL, PL/SQL, mySQL
  - Cursoare, proceduri stocate, functii, triggere
- Limbajul SQL integrat (Embeded SQL)
- Interfete de programare a aplicatiilor de baze de date
  - Interfata ODBC
  - Interfata JDBC

Dezvoltarea sistemelor de baze de date (1)

- ***Sistemul informatic*** *(information system)* al unei organizatii include toate resursele acelei organizații care sunt implicate în colectarea, administrarea, utilizarea şi diseminarea informațiilor
- Sistemele informatice:
  - Pana in anii 1970 erau sisteme de fişiere (pe disc sau bandă magnetică)
  - Actual se folosesc sisteme de baze de date, care permit gestionarea unor volume de date mari într-un timp redus, cu protecția si securitatea datelor
- Fazele de dezvoltare a sistemelor de baze de date:
  - *Analiza şi definirea sistemului*: definirea scopului sistemului de baze de date, a utilizatorilor şi a aplicațiilor acestuia
  - *Proiectarea sistemului*: în această etapă se realizează proiectul sistemului, pentru un anumit SGBD ales
  - *Implementarea*: este etapa în care se scriu definițiile obiectelor bazei de date (tabele, vederi, etc.), se implementează aplicațiile software și se introduc datele
  - *Testarea şi validarea*: noul sistem de baze de date este populat cu date, testat şi validat cât mai riguros posibil
- În mod tipic, dezvoltarea unui sistem de baze de date constă din:
  - Dezvoltarea structurii si a continutului bazei de date
  - Dezvoltarea modulelor de prelucrare a datelor (tranzactii)

Dezvoltarea sistemelor de baze de date (2)

- Fazele importante de dezvoltare a sistemelor de baze de date sunt:

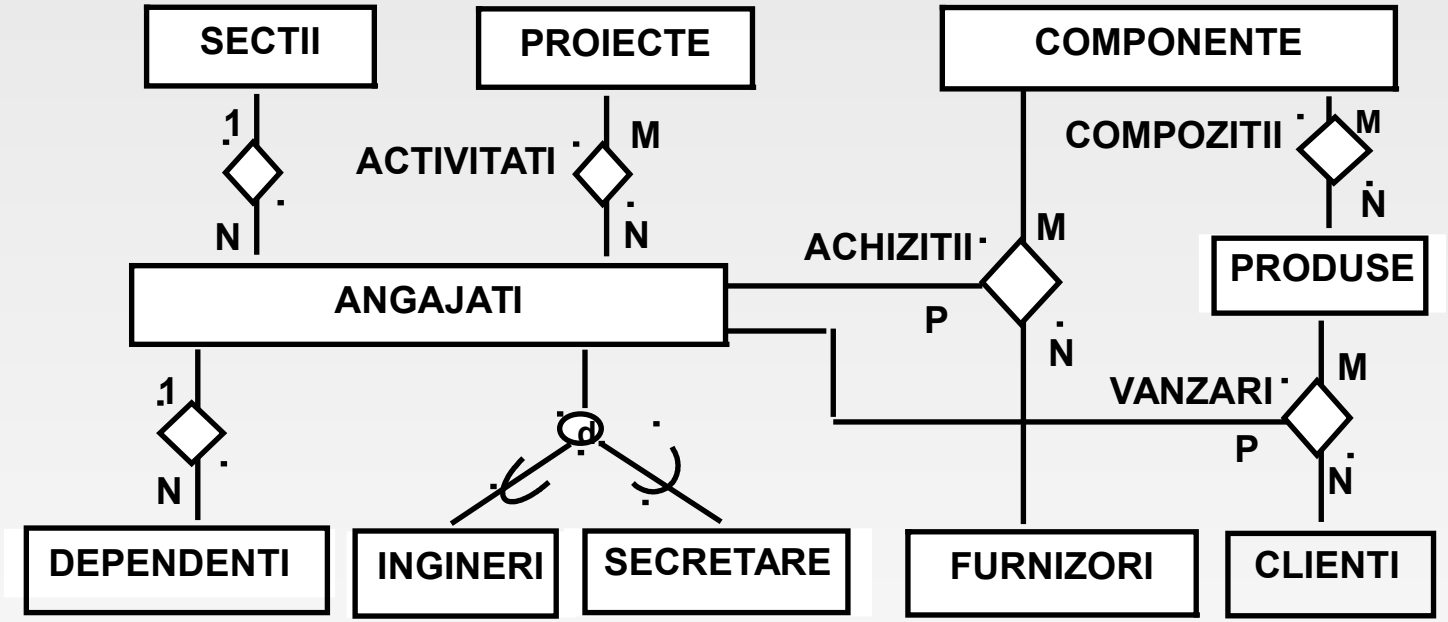


Colectarea si analiza cerințelor

- ***Colectarea și analiza cerințelo**r*: pentru proiectare este necesar să se știe:
  - ce rezultate se aşteaptă utilizatorii potențiali să obțină de la baza de date respectivă
  - ce informații primare sunt disponibile pentru acestea
  - ce aplicații se vor executa (aplicații de gestiune a stocurilor, aplicații contabile, aplicații de urmărire a consumurilor, aplicații de salarizare, etc.).
- Toate acestea sunt informații slab structurate, în general în limbaj natural, pe baza cărora se pot construi diagrame, tabele, grafice etc., manual sau folosind diferite instrumente software de proiectare
- Dar din aceste informatii trebuie sa fie extrase date precise de proiectare a bazelor de date si a aplicatiilor
- Această fază este puternic consumatoare de timp, dar este crucială pentru succesul sistemului informatic
- ***Proiectarea conceptuală a bazei de date*:**
  - Proiectarea schemei conceptuale de nivel inalt a bazei de date
  - Proiectarea conceptuală a tranzactiilor
- Proiectul conceptual de nivel înalt este independent de modelul de date și de SGBD și reprezintă o descriere principială și stabilă a bazei de date, care poate fi utilizată pentru particularizarea pentru orice model de date sau SGBD
- Se poate reprezenta prin diagrame E/A sau în limbajul UML

Exemplu: proiectarea conceptuală a unei baze de date

- Baza de date INTREPRINDERE cu multimile de entitati puternice:
  - SECTII(Nume,Buget)
  - ANGAJATI(Nume, Prenume, DataNasterii, Adresa, Functie, Salariu)
  - PROIECTE(Denumire,Termen, Buget)
  - PRODUSE(Denumire, Descriere)
  - COMPONENTE(Denumire, Descriere)
  - FURNIZORI(Nume,Prenume,Adresa) ; CLIENTI(Nume, Prenume, Adresa)
- Diagrama E-A:



Alegerea unui model de date și a unui SGBD

- Alegerea modelului de date - în funcție de cerințele aplicației (complexitatea datelor, a interogărilor): ierarhic, rețea, relațional, obiect-orientat, obiect-relațional
- Factori pentru alegerea unui SGBD în cadrul unui model de date ales: tehnici, economici şi administrativi
- Factorii tehnici:
  - Capacitatea de stocare a datelor
  - Posibilitățile de control al concurenței şi de refacere a datelor
  - Configurația hardware-software necesară
  - Cerințe de dezvoltare a programelor de aplicații (limbaje, compilatoare etc.)
- Factorii economici şi administrativi:
  - Costul de achiziție a software-ului, care include costul de bază, la care se adaugă diferite opțiuni (opțiuni de salvare-refacere, documentație, limbaje şi interfețe de programare, drivere, etc.)
  - Costul de întreținere, pentru a obține serviciul furnizorului de menținere la zi a SGBD
  - Costul de pregătire a personalului se referă la cursurile care se organizează pentru persoanele care se ocupă cu întreținerea şi operarea sistemului
  - Cunoştințele de programare a personalului într-un anumit SGBD
- Beneficiile achiziționării unui anumit SGBD nu sunt uşor de apreciat sau de măsurat, dar analiza raportului cost-performanțe poate da o imagine a rezultatelor obținute

Proiectarea logică a bazelor de date relaționale

- Se porneste de la schema conceptuală de nivel înalt (diagrama E-A), si se realizează schema logica (diagrama logica) a bazei de date relaționale
- Proiectarea logică a unei baze de date relaționale poate fi realizată în două faze:
  - Transpunerea schemei conceptuale în schemă logică independentă de SGBD; acestă fază implică:
    - Proiectarea relațiilor corespunzătoare mulțimilor de entități din diagrama E-A
    - Proiectarea asocierilor între relații corespunzătoare asocierilor între mulțimile de entitități
  - Rafinarea schemei logice pentru un anumit SGBD ales (modul de generare a cheilor primare, definirea constrângerilor, etc.)
- În mod frecvent, etapele de proiectare conceptuala si proiectare logica (cu cele două faze) se efectueaza impreuna, proiectand direct schema (diagrama) logica a bazei de date cu ajutorul toolset-urilor de dezvoltare oferite de SGBD-ul utilizat

Diagrama logică a bazei de date INTREPRINDERE (1)

- Dezvoltata in Microsoft Access

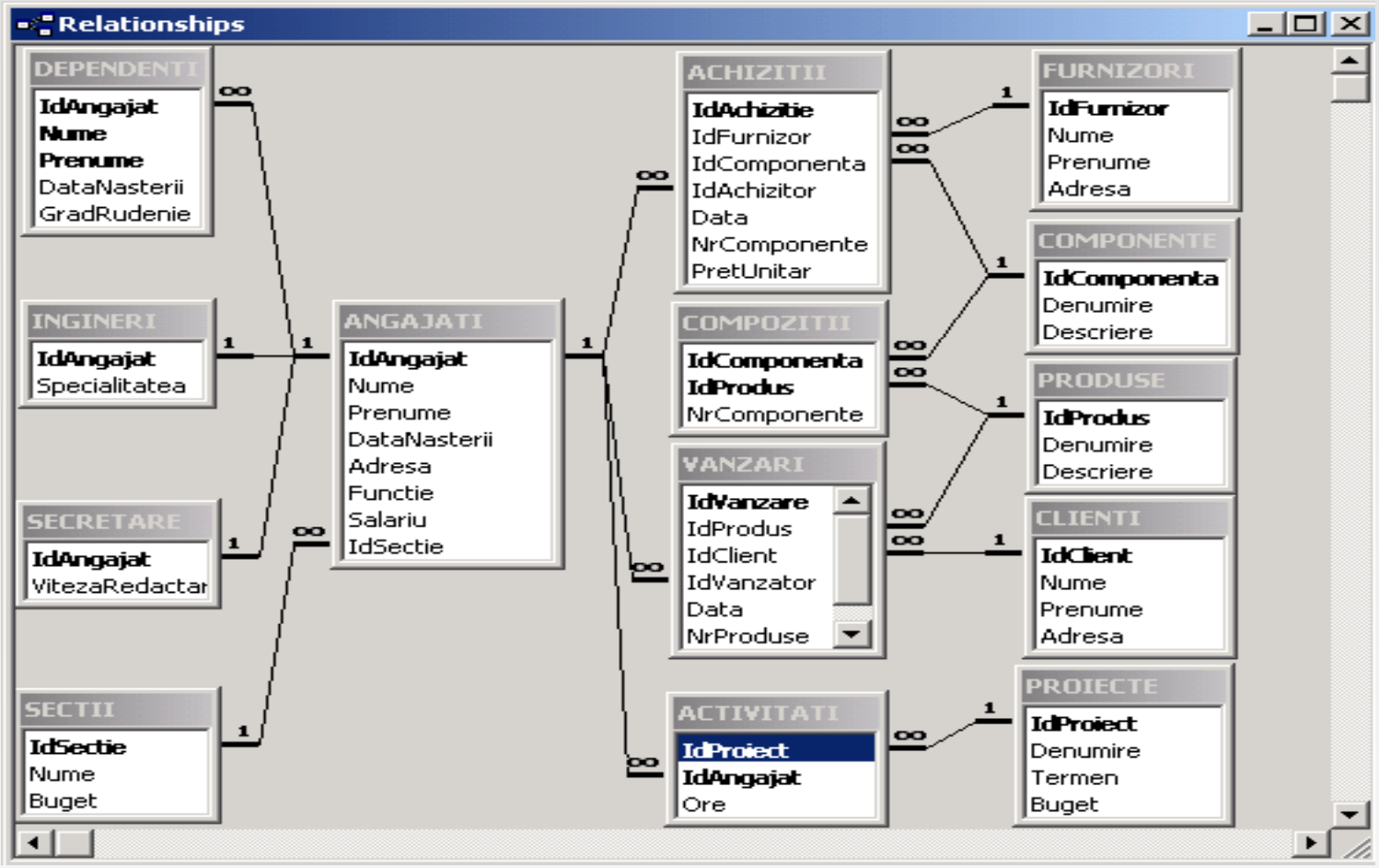
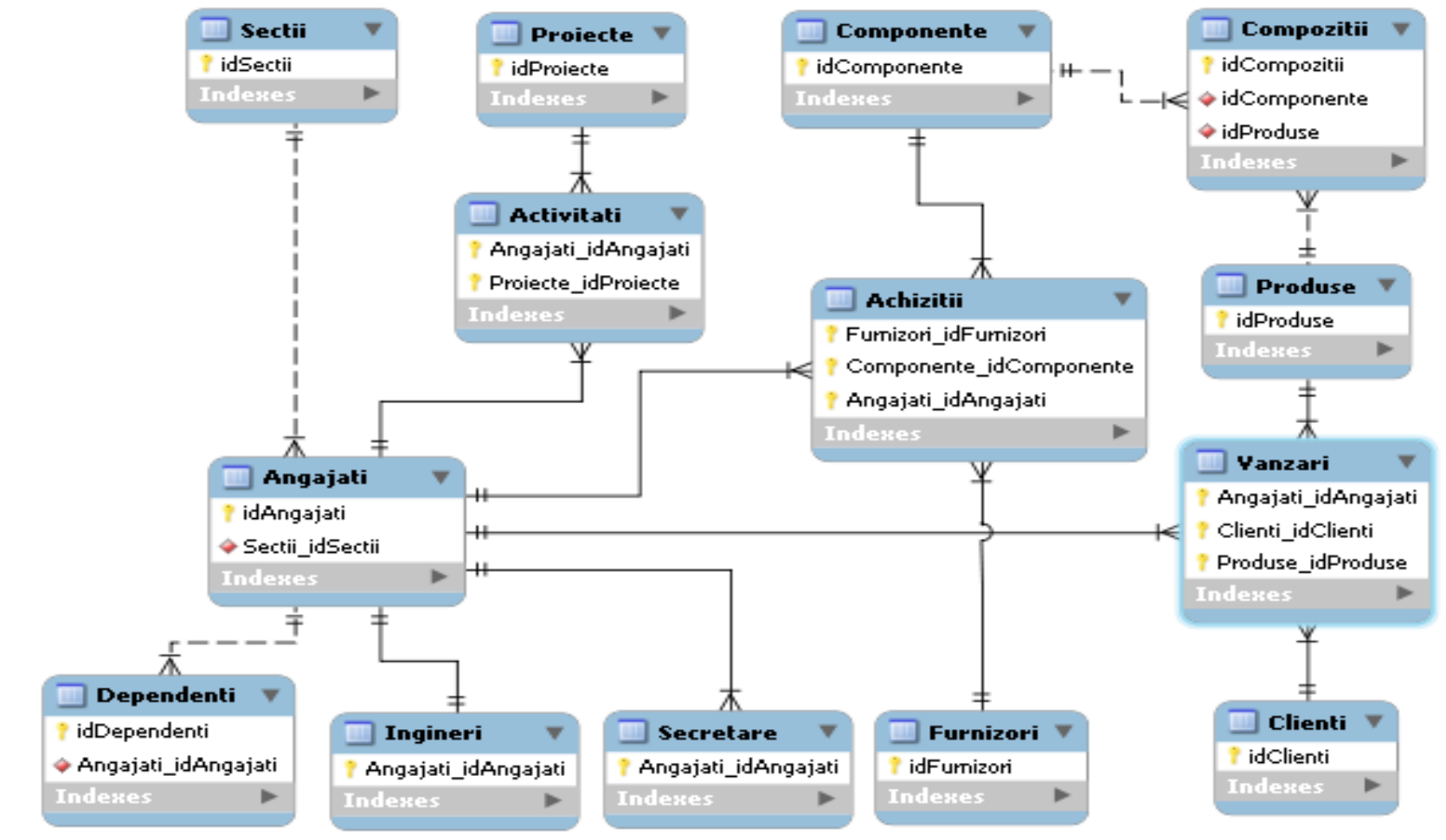


Diagrama logică a bazei de date INTREPRINDERE (2)

- Dezvoltata in MySQL Workbench



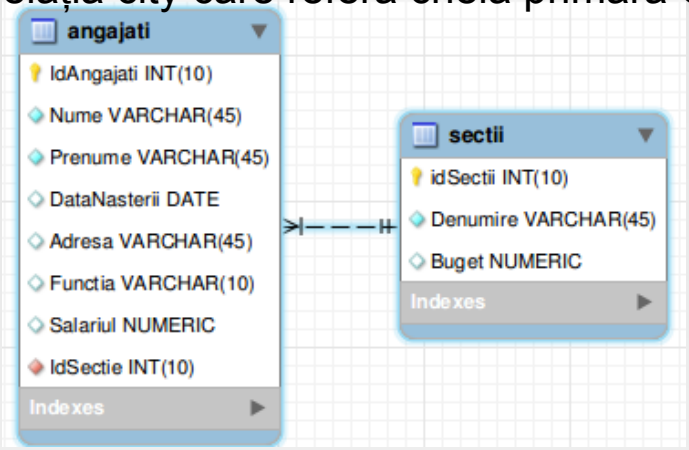
Proiectarea relatiilor

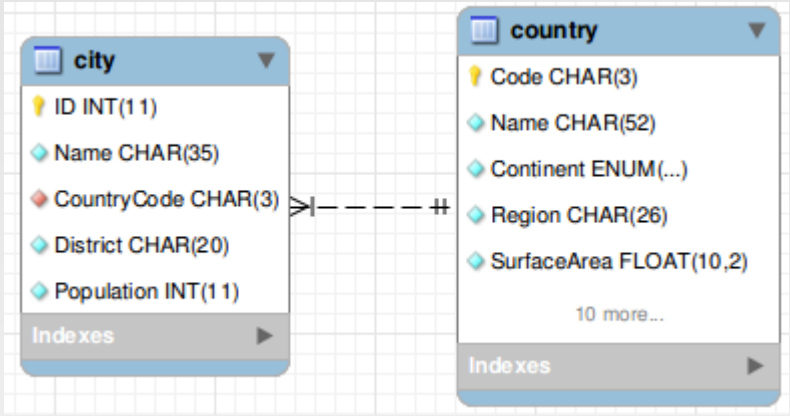
- Mulțimile de entități din diagrama E-A devin relații:
  - Numele fiecărei relații trebuie să fie unic în baza de date
  - Atributele relatiei corespund atributelor entităților din multimea de entitati data
  - Cheia primară se defineşte:
    - fie ca o cheie primara naturală (combinație de atribute care definesc în mod unic un tuplu)
    - fie ca o cheie primară artificială
    - De exemplu, în relațiile ANGAJATI, SECTII, PROIECTE, COMPONENTE, PRODUSE, FURNIZORI, CLIENTI s-a adăugat câte o cheie primară artificială (IdAngajat, IdSectie etc.)
- Mulțimile de entități slabe din diagrama E-A devin relații aflate în asociere N:1 cu relația corespunzătoare mulțimii de entități de care acestea depind
  - Pentru realizarea acestei asocieri, în relația dependentă se adaugă o cheie străină care referă cheia primară a relației puternice referite; de exemplu: cheia straina dAngajat in introdusa in relatia DEPENDENTI . De ex:

    DEPENDENTI(I*dAngajat*, Nume, Prenume, DataNasterii, GradRudenie)
  - Cheia primară a relației dependente poate fi:
    - o combinație formată din atributul cheie străină şi alte atribute care asigură posibilitatea de identificare unică a unui tuplu sau poate fi o cheie artificială. De ex: (IdAngajat, Nume, Prenume)
    - sau o cheie primara artificiala (de ex. idDependenti)

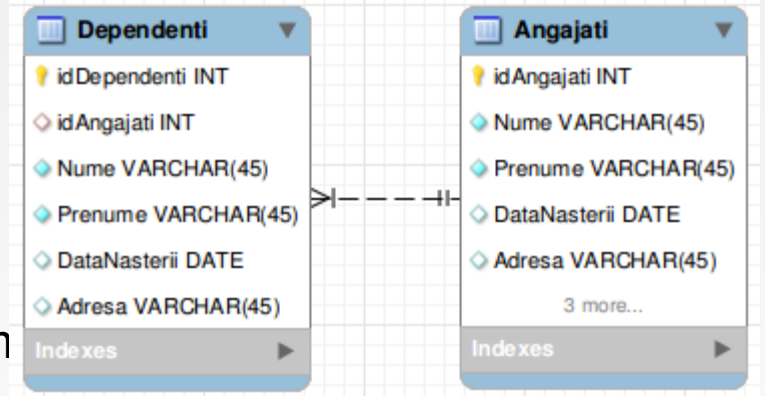
Proiectarea asocierilor binare N:1

- ***Asocierea binară N:1*** dintre două mulțimi de entități puternice se realizează prin intermediul unei chei străine introdusă în prima relație (cea cu multiplicitatea N a asocierii) care referă cheia primară din cea de-a doua relație; exemple:
  - Asocierea N:1 între relațiile ANGAJATI - SECTII se realizează prin cheia străină IdSectie din relatia ANGAJATI care referă cheia primară IdSectie din relația SECTII
  - Asocierea N:1 între relațiile city-country se realizează prin cheia straina CountryCode din relația city care referă cheia primară Code din relația country



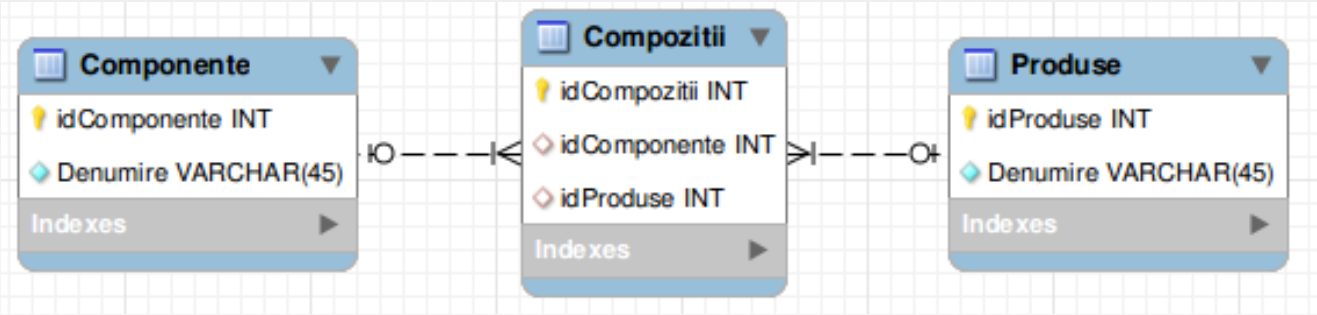


- Asocierea binară N:1 dintre o mulțime de entități slabe si mulțimea de entități puternice de care depinde se realizeaza printr-o cheie straina în relația multimii de entitati slabe care referă cheia primară a relației mulțimii de entități puternice
  - Exemplu: asocierea N:1 între relațiile Dependenti si Angajati se realizează prin cheia straina IdAngajat din relatia Dependenti care referă cheia primară IdAngajat din relația Angajati



Proiectarea asocierilor binare M:N

- O asociere binară M:N dintre două mulțimi de entități se realizeaza cu o noua relație, numită relație de asociere
- Aceasta are rapoartele de cardinalitate M :1, respectiv N :1 cu fiecare din cele două relații date, prin intermediul a două chei străine care referă cheile primare din fiecare din cele două relații asociate
- Cheia primară a unei relații de asociere poate fi:
  - o cheie primara artificială
  - sau poate fi compusă din cheile străine, eventual împreună cu alte atribute ale relației
- Exemplu: asocierea M:N dintre relațiile COMPONENTE-PRODUSE se realizeaza cu o relație de asociere, numită COMPOZITII
  - Relatia COMPOZITII contine cheile străine IdComponenta şi IdProdus care referă cheile primare IdComponenta, IdProdus ale relațiilor Componente, respectiv Produse
  - Cheia primară a relației COMPOZITII este o cheie primară artificiala (IdCompozitii)



Alt exemplu de asociere M:N

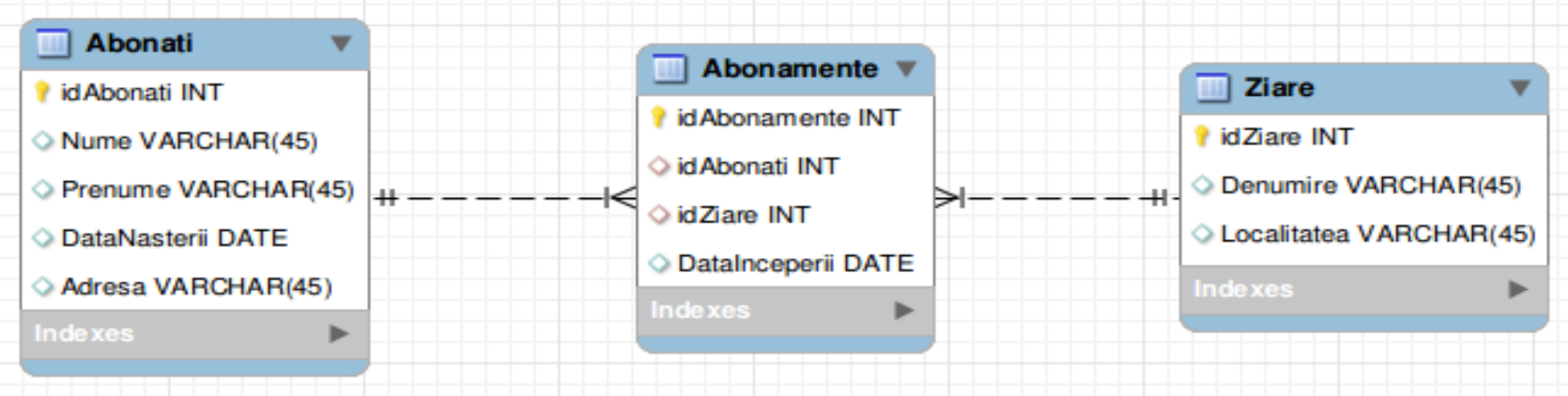
- Fie relațiile Abonati, Ziare, cu asocierea M:N între ele, realizată prin relația suplimentară de asociere Abonamente
  - Aceasta are rapoartele de cardinalitate M :1, respectiv N :1 cu fiecare din cele două relații date, prin intermediul a două chei străine care referă cheile primare din relațiile asociate
- Interogarea “Care sunt abonatii si abonamentele lor” se rezolvă în relația de asociere, dacă nu ne intereseza decât cheile abonaților si ale ziarelor

  Alg. relațională: $q = \prod_{idAbonati,idZiare} (Abonamente)$

  SQL: SELECT idAbonati, idZiare FROM Abonamente;
- Informații suplimentare despre abonați (Nume, Prenume) și ziarele la care aceștia sunt abonați (Denumire) se obțin prin joncțiunea relației Abonamente cu fiecare din cele două relații:

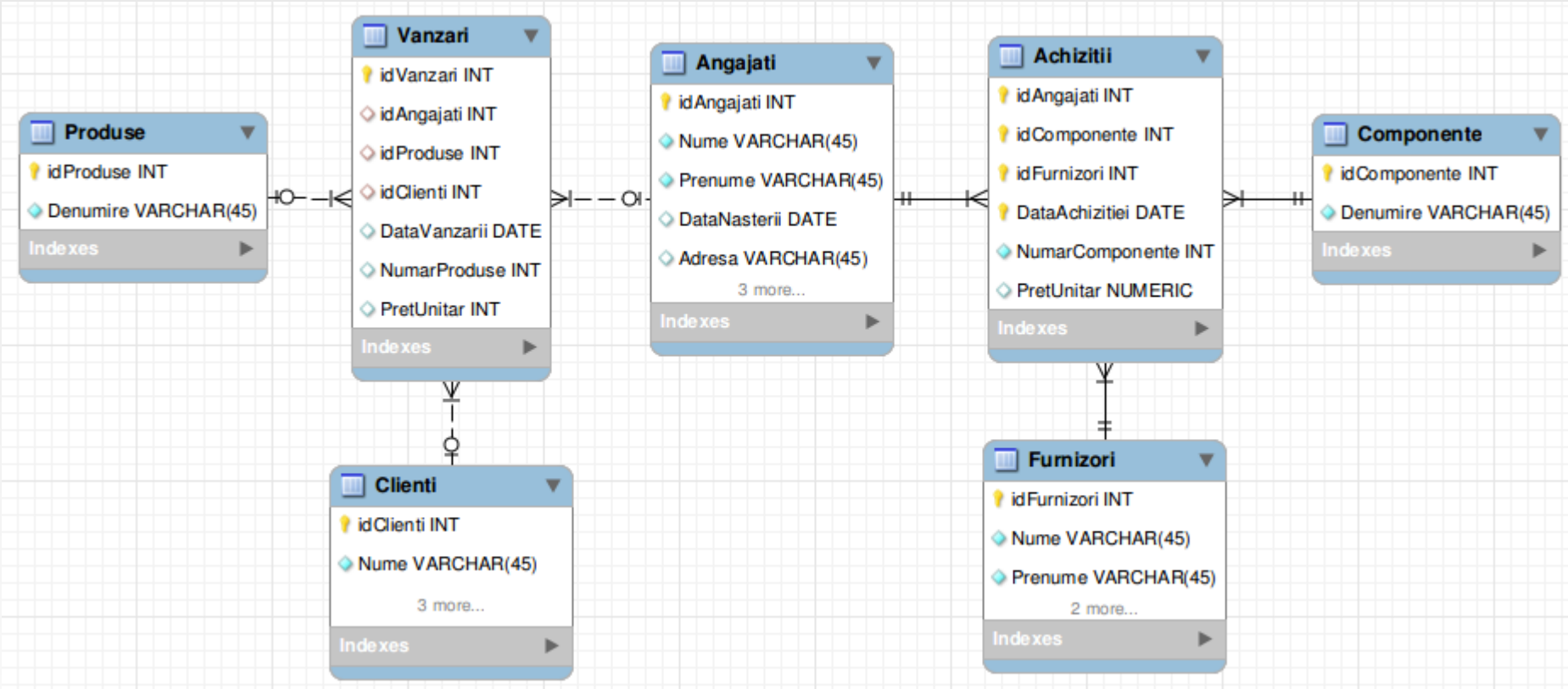
  Alg. Relațională: $q = \prod_{Nume,Prenume,\ Denumire} (Abonati \bowtie Abonamente \bowtie Ziare)$

  SQL: SELECT Nune, Prenume, Denumire FROM Abonati, Abonamente, Ziare
  WHERE Abonati.idAbonati Abonamente.idAbonati AND Ziare.idZiare=Abonamente.idZiare



Proiectarea asocierilor multiple M:N:P...

- ***Asocierea multiplă M:N:P:....*** se realizează la fel ca asocierea binară M:N, prin intermediul unei relații de asociere care se află în asociere M:1, N:1, P:1, etc, cu fiecare din relațiile date, prin câte o chei străină care refera cheia primară coresp.
- Cheia primară a unei relații de asociere multiplă poate fi:
  - O cheie primara artificială; ex.(a): în relația **Vanzari**, este definită cheia primară artificială idVanzari
  - Cheia primară poate fi compusă din toate cheile străine, eventual împreună cu alte atribute; ex. (b); in relația de asociere **Achizitii** cheia primară este formată din toate cheile străine plus atributul DataAchizitiei



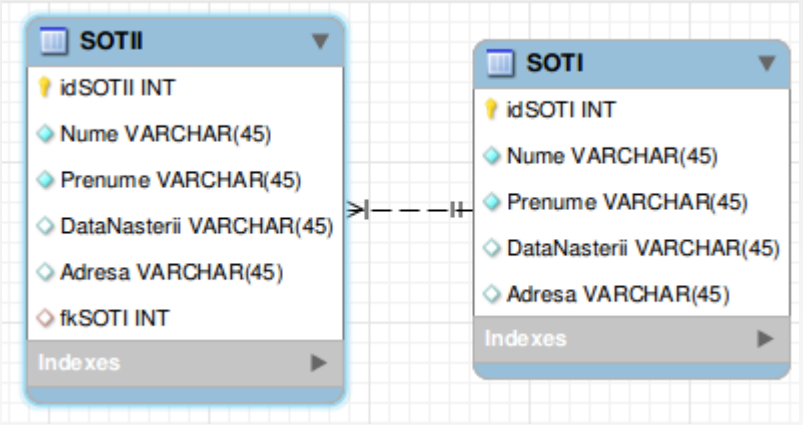
Proiectarea asocierilor binare 1:1 (1)

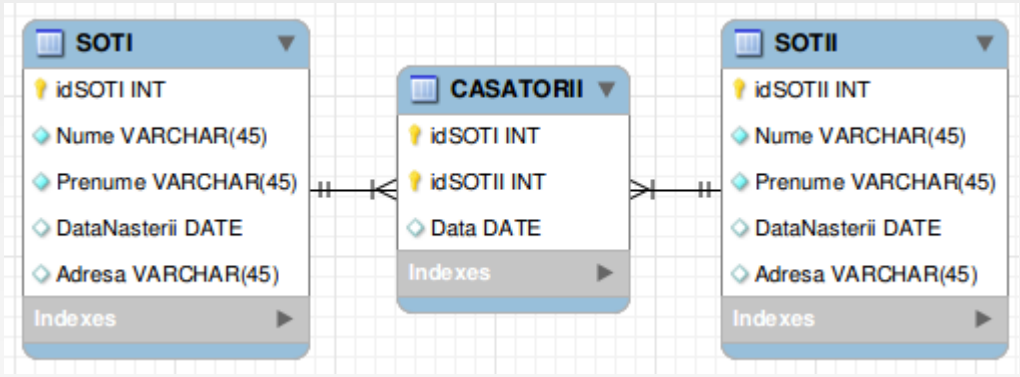
- ***Asocierea binară 1:1 între două mulțimi de entități puternice*** se poate transpune în modelul relațional în două moduri:
  - fie prin intermediul unei chei străine (caz particular al asocierii 1:N)
  - fie printr-o relație de asociere (caz particular al unei asocierii M:N).
- Exemplu: intre relațiile SOTI, SOTII, se realizeaza o asociere 1:1 cu o cheie straina introdusa intr-una dintre relatii:

  SOTI (<u>idSOTI</u>, Nume, Prenume, DataNasterii, Adresa)

  SOTII (<u>idSOTII</u>,Nume, Prenume, DataNasterii, Adresa, *fkSOTI*)
- Realizarea asocierii 1:1 între două relații folosind o relatie de asociere

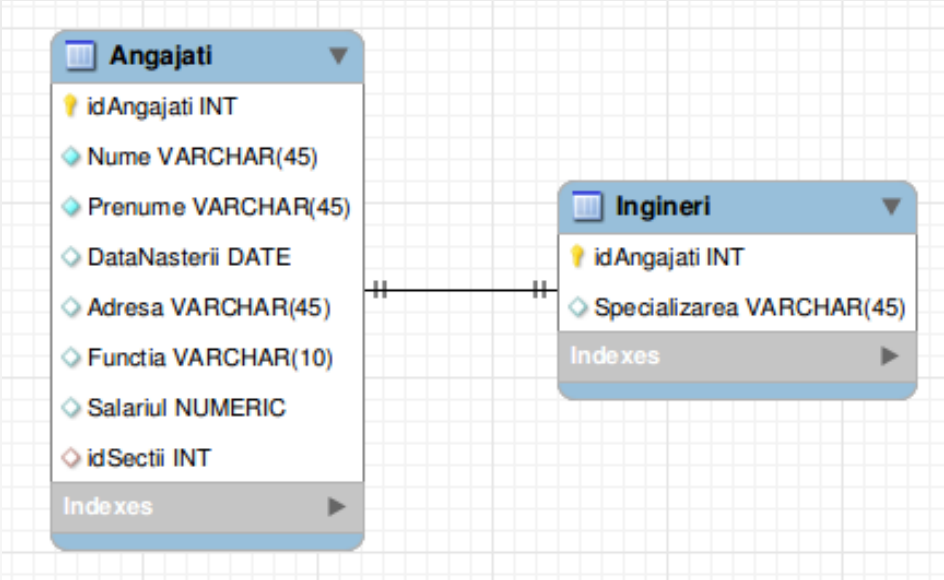
  De exemplu: CASATORII(*<u>idSOTI</u>*, *<u>idSOTII</u>*, DataCasatoriei)
- În ambele soluții, SGBD-ul nu limitează raportul de cardinal. strict la 1:1, ci acceptă asocieri 1:N, respectiv M:N; limitarea trebuie asigurată de aplicație





Proiectarea asocierilor binare 1:1 (2)

- ***Asocierea binară 1:1 dintre o mulțime de entități de subtip şi mulțimea de entități supertip*** se poate transpune în modelul relațional prin definirea în relația corespunzătoare subtipului a unei chei străine, care referă cheia primară din relația se supertip și este în acelaşi timp şi cheie primară
  - De exemplu: asocierea 1:1 între relația de subtip INGINERI și relația supertipului corespunzător (ANGAJATI) se realizează prin cheia primară IdAngajati din INGINERI care este și cheie străină și referă cheia primară IdAngajati din relația ANGAJATI
  - În acest caz, o entitate de subtip este reprezentată prin două tupluri: un tuplu în relația de subtip (INGINERI) care conține valoarea atributului Specializarea și tuplul referit din relația de supertip (ANGAJATI) care vonține valorile tuturor celorlalte atribute (Nume, Prenume, DataNasterii, Adresa etc.)
  - Asocierea este în mod real 1:1, vedificată de SGBD prin verificarea cheiilor



Rezumat: proiectarea logică a bazelor de date

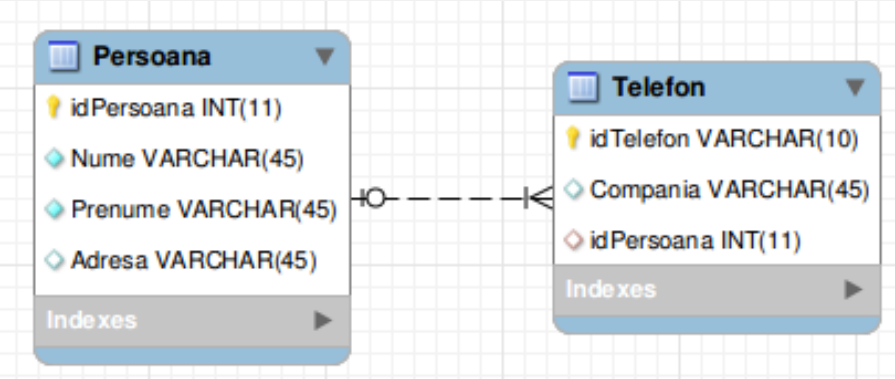
- ***O mulțime de entități*** se reprezintă printr-o relație
- ***Asocierea M:N sau M:N:P*... se reprezintă printr-o relație de asociere**
- ***Asocierea N:1*** se reprezintă corect, cu respectarea raportului de cardinalitate, printr-o cheie străină introdusă în relația cu multiplicitatea N (care referă)
  - Asocierea N:1 se poate reprezenta și printr-o relație de asociere, dar, în acest caz, se poate să nu fie respectat raportul de cardin. N:1 dacă nu se prevăd funcții speciale în programul de aplicație, ceea ce este costisitor ca efort de progr. și timp de execuție
- ***Asocierea 1:1 între două mulțimi de entități puternice*** se reprezintă fie printr-o cheie străină, (ca un caz particular al asocierii N:1) fie printr-o relație de asociere (ca un caz particular al asocierii M:N)
  - În ambele cazuri se poate să nu fie respectat strict raportul de cardinaliate 1:1, dacă nu se prevăd funcții speciale în programul de aplicație
- ***Asocierea 1:1 între o mulțime de subtip și o mulțime de supertip*** se reprezintă corect printr-o cheie stăină în relația de subtip (care referă cheia primară din relația supertip) și este și cheie primară
  - În orice altă reprezentare (printr-o cheie străină în relația de subtip, dar care să nu fie și cheie primară, sau printr-o relație de asociere) se poate să nu fie respectat strict raportul de cardinalitate 1:1, dacă nu se prevăd funcții speciale în programele de aplicație
- ***În concluzie, în modelul relațional toate asocierile folosesc chei străine***

Proiectarea relațiilor normalizate

- ***O formă normală a unei relații*** (*normal form*) impune respectarea unor anumite condiții de către valorile atributelor şi dependențele de date definite pe acea relație
- ***Dependențele de date*** (data dependencies) reprezintă constrângeri care se impun valorilor atributelor unei relații şi care determină proprietățile relației în raport cu operațiile de inserare, ştergere şi actualizare a tuplurilor.
- Dependentele de date:
  - Dependente functionale: E.F. Codd a propus trei forme normale: FN1, FN2, FN3; apoi a fost introdusă forma normală Boyce-Codd (FNBC)
  - Dependențelor multivalorice: forma normala 4 (FN4)
  - Dependențelor de joncțiune: forma normala 5 (FN5)
- Formele normale ale relațiilor formează o colecție ordonată (FN1, FN2, FN3, FNBC, FN4, FN5), şi ele impun condiții din ce în ce mai restrictive asupra dependențelor de date
- Ordonarea formelor normale de la FN1 la FN5 înseamnă că orice relație aflată în FN2 este în FN1, orice relație în FN3 este în FN1 şi FN2 etc.
- ***Normalizarea relațiilor*** (*normalization*) constă în descompunerea lor, astfel încât relațiile să fie in forme normale cât mai avansate

Forma normală 1 (FN1)

- O relație este normalizată în prima formă normală (FN1) dacă fiecare atribut ia numai valori atomice şi scalare din domeniul său de definiție
- O situație frecvent întâlnită este aceea în care un atribut poate lua mai multe valori pentru fiecare entitate
  - De exemplu, în mulțimea de entități PERSOANE (Nume, Prenume, Adresa, NrTelefon) atributul NrTelefon poate lua mai multe valori (numărul telefonului de acasă, al telefonului de la birou, al telefonului mobil, etc)
- Relația în care un atribut poate avea valori multiple (un vector de valori) este o relație nenormalizată, care nu este admisă de SGBD relaționale
- Transformarea unei relații nenormalizate în relație normalizată în prima forma normală (FN1):
  - Se înlocuieşte atributul care ar putea avea valori multiple cu câte un atribut pentru fiecare din posibilitățile existente: Exemplu: PERSOANA (IdPersoana, Nume, Prenume, Adresa,TelefonAcasa, TelefonBirou, TelefonMobil)
  - Se înlocuieşte atributul care ar putea avea valori multiple cu o nouă relație care referă relația inițială printr-o cheie străină. Exemplu: PERSOANA (IdPersoana, Nume, Prenume, Adresa), TELEFON (NrTelefon, *IdPersoana,* Descriere)



Dependențe de date

- ***Dependențele funcționale*** (*functional dependencies*) sunt constrangeri in relatii, prin care valoarea unui anumit set de atribute determina in mod unic valoarea altor atribute
- Fie R o schema de relatie si doua submultimi ale atributelor sale: $X \subseteq R$, $Y \subseteq R$ Dependenta functionala (DF) $X \rightarrow Y$ exista in R daca si numai daca pentru orice stare a relatiei r(R), egalitatea valorilor atributelor X din doua tupluri t_1 si t_2 din r ($t_1 \in$ r si $t_2 \in$ r) implica egalitatea valorilor atributelor Y din acele tupluri, adică:

  $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$
- ***O dependență multivalorică*** - DMV- (multivalued dependency) $X \rightarrow\rightarrow Y$ specificată pe schema de relatie R = {X,Y,Z} stabileşte urm. constrângeri pe care trebuie să le respecte orice relație r(R): dacă există două tupluri t1 şi t2 în r astfel ca t1[X] = t2[X] = x, atunci vor exista şi tuplurile t3 şi t4 cu urm proprietăți:

  t3[X] = t4[X] = t1[X] = t2[X] = x;

  t3[Y] = t1[Y] = y1 *şi* t4[Y] = t2[Y] = y2 ;

  t3[Z] = t2[Z] = z2 *şi* t4[Z] = t1[Z] = z1 .
- ***Dependențele de joncțiune****: Fie o schemă de relație* R *şi* R1, R2, ...Rk *submulțimi de atribute ale lui* R*, unde* $R1 \cup R2 \cup \ldots \cup Rk = R$*. Se spune că există o dependență de joncțiune (DJ) pe* R*, notată* *(R1,R2,...Rk)*, dacă şi numai dacă orice relație* r(R) *este egală cu joncțiunea proiecțiilor relației pe submulțimile* R1,R2,..Rk*, adică* $r = \Pi_{R1}(r) >< \Pi_{R2}(r) \ldots >< \Pi_{Rk}(r)$

Condițiile de normalizare a relațiilor

- Conditiile de normalizare in FNBC, FN4 si FN5 se pot rezuma la faptul că într-o relație normalizată nu există decât dependențe determinate de chei:
  - O relație este în FNBC dacă orice DF este determinată de o cheie a relației
  - O relație este în FN4 dacă orice DF sau DMV este determinată de o cheie a relației
  - O relație este în FN5 dacă orice DF, DMV sau DJ este determ. de o cheie a relației
- O relație într-o formă normală redusă prezintă redundanțe și anomalii de actualizare a datelor
  - Cu cât gradul de normalizare este mai ridicat, cu atât există mai puține redundanțe și anomalii de actualizare
  - Relațiile complet normalizate (în FN5) nu au redundanțe sau anomalii de actualizare
- Ex. relație în FN1: AP = {IdAngajat, Nume, Prenume, Adresa, IdProiect, Ore} cu cheia primară PK = {IdAngajat, IdProiect}

IdAngajat	Nume	Prenume	Adresa	*IdProiect*	Ore
1	Ionescu	Ion	Bucuresti	P1	100
2	Popescu	Petre	Ploiesti	P1	80
3	Marinescu	Marin	Craiova	P1	200
1	Ionescu	Ion	Bucuresti	P2	100
2	Popescu	Petre	Ploiesti	P2	120

Redundanța datelor și anomaliile de actualizare

- Relatia AP se află într-o formă normală redusă (FN1): DF: IdAngajat → Nume, IdAngajat → Prenume, IdAngajat → Adresa nu sunt determinate de o cheie
- De aceea există valori redundante ale atributelor (Nume, Prenume, Adresa) și există mai multe anomalii de actualizare, inserare, ștergere a relațiilor:
  - ***Anomalii de actualizare:*** *dacă se modifică într-un tuplu valoarea unuia din atributele care au valori redundante, starea relației devene inconsistentă*
  - ***Anomalii de inserare***: nu se pot introduce date despre un angajat (numele, prenumele, adresa) dacă nu există cel puțin un proiect la care acesta să lucreze
  - ***Anomalii de ştergere***: dacă se şterg toate tuplurile ref. la un anumit proiect, se pierd toate datele despre acei angajați care lucrează doar la proiectul respectiv
- Eliminarea anomaliilor provocate de redundanta datelor se poate face:
  - Fie prin prevederea unor proceduri stocate (sau triggere) care sa verifice corectitudinea fiecarei operatii de actualizare a relatiilor; de exemplu, adresa unui angajat să se modifice în toate tuplurile în care apare angajatul respectiv
  - Fie prin normalizare, care se face prin descomp unei relatii slab normalizate, care prezinta redundante și anomalii, in relati normalizate, în care nu mai există redundanțe și anomalii
  - Exemplu: descomp. relatiei AP(care este în FN1) in relatiile, A si P (care sunt în FN5)

A

IdAngajat	Nume	Prenume	Adresa
1	Ionescu	Ion	Bucuresti
2	Popescu	Petre	Ploiesti
3	Marinescu	Marin	Craiova

P

IdAngajat	IdProiect	Ore
1	P1	100
2	P1	80
3	P1	200
1	P2	100
2	P2	120

Normalizarea relațiilor (1)

- Așadar normalizarea relațiilor se realizează prin descompunerea lor
- Două proprietăți ale unei descompuneri sunt importante: proprietatea de conservare a informației și proprietatea de conservare a dependențelor
  - ***Proprietatea unei descompuneri de conservare a informației*** înseamnă că prin joncțiunea relațiilor rezultate se obține o relație identică cu relația inițială (o astfel de desc. se mai numește și descompunere fără pierdere de informație la joncțiune)
  - ***Proprietatea unei descompuneri de conservare a dependențelor*** înseamnă că reuniunea mulțimilor dependențelor din relațiile rezultate să fie echivalentă cu mulțimea depenențelor din relația inițială
  - O descompunere care are ambele proprietăți se numește descompunere reversibilă
- Pentru normalizare nu se admit decât descompuneri care conservă informația, dar se pot admite descompuneri care nu conservă dependențele
  - S-a demonstrat si exista algoritmi prin care orice relatie poate fi descompusă reversibil (cu conservarea informatiei si conservarea dependențelor) in relatii in FN2 sau FN3
  - S-a demonstrat si exista algoritmi prin care orice relatie poate fi descompusă in relații FNBC, FN4 sau FN5 fara pierdere de informatie la jonct., dar se pot pierde unele dependențe; în acest caz dependențele pierdute prin descompunere se impun prin trigere sau proceduri stocate

Normalizarea relațiilor (2)

- În exemplul dat, relația AP este slab normalizată (FN1), iar relatiile rezultate A si P sunt complet normalizate (FN5), nu mai au redundanțe sau anomalii
- Interogarile care necesită joncțiunea dintre A și P sunt mai ineficiente decat interogarea coresp. în relația AP, deoarece joncțiunea este o operație costisitoare

  Exemplu: "Care este numărul de ore lucrate de Ionescu la proiectul P1 ?":

  Q1 = $\Pi_{\text{Ore}}\ \sigma_{\text{Nume ='Ionescu' AND IdProiect ='P1'}}$ (AP) Q2 = $\Pi_{\text{Ore}}\ \sigma_{\text{Nume ='Ionescu' AND IdProiect ='P1'}}$ (A ⊳⊲ P)
- De aceea, în general, se recomandă ca:
  - Relațiile asupra cărora se efectuează operații de actualizare frecvente să fie normalizate într-o formă normală cât mai avansată
  - Relațiile asupra cărora se efectuează interogări frecvente pot fi păstrate într-o formă de normalizare mai redusă, dar trebuie să fie prevăzute proceduri pentru impunerea constrângerilor explicite (dependențe care nu sunt determinate de cheile relației)
- În cadrul proiectării:
  - Relațiile care modelează un singur tip de entitate sau de asociere, tind să fie într-un grad înalt de normalizare (ex. A modelează "angajati", P modelează asocierea angajati-proiecte)
  - Relațiile care amestecă informații despre mai multe tipuri de entități și asocieri tind să fie într-un nivel scăzut de norm.alizare; exemple:
    - Relatia AP modeleaza o multime de entitati (angajati) si o asociere (asocierea angajatilor cu proiectele) si de aceea prezinta dependente functionale
    - O relatie care modeleaza doua asocieri M:N indelendente prezinta dependente multivalorice (DMV)
    - O relatie care modeleaza mai mult de doua asocieri M:N independente prezinta dependente de jonctiune

Normalizarea relațiilor (3)

- Analiza normalizării relațiilor trebuie să fie facuta pentru orice proiect de baze de date, pentru a asigura funcționarea corectă a acesteia:
  - Dacă o relație se păstrează într-o formă de normalizare mai redusă, atunci trebuie să se prevadă proceduri de verificare şi de evitare a anomaliilor prin impunerea dependențelor care nu sunt determinate de cheile relației (ca si constrângeri explicite)
  - Dacă se normalizeaza o relație, dar prin descompunere se pierd unele DF, acestea pot fi impuse explicit prin proceduri stocate sau funcții
  - Normalizarea relațiilor asigură un proiect al bazei de date mai concis şi de aceea se consideră că a normaliza este avantajos, chiar dacă normalizarea nu este o garanție că s-a realizat cel mai bun model
- Dar, cu cât nivelul de normalizare creşte, cu atât se obțin mai multe relații cu grad mai mic şi pentru fiecare interogare sunt necesare mai multe operații de joncțiune, ceea ce face ca timpul de execuție a interogărilor să crească

Proiectarea fizică a bazelor de date

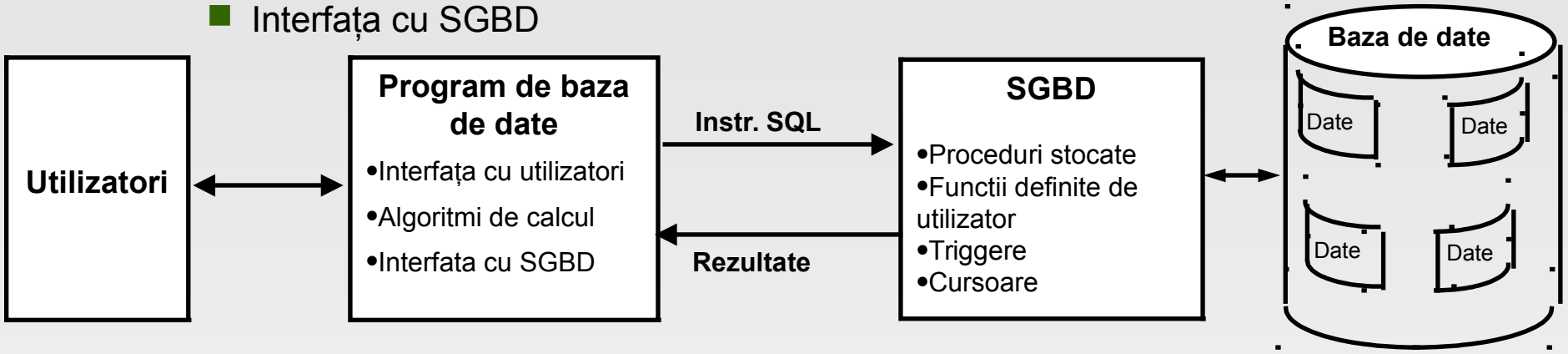
- Fiecare SGBD oferă o mai multe de opțiuni de organizare a fişierelor şi a modului de acces la datele stocate:
  - Indexuri
  - Tipuri de fisiere
  - Gruparea înregistrărilor corelate în aceleaşi blocuri pe disc (clustering)
- Proiectarea fizică a bazei de date este procesul de alegere a acestor structuri de memorare şi de acces la fişierele bazei de date, pentru a obține performanțe cât mai bune pentru SGBD-ul ales, pentru cât mai multe din aplicațiile proiectate
- Parametrii generali de alegere a opțiunilor proiectului fizic al bazei de date:
  - Timpul de răspuns: intervalul de timp dintre lansarea în execuție a unei tranzacții şi primirea răspunsului la acea tranzacții
  - Capacitatea tranzacțională (transaction throughput): numărul mediu de tranzacții care pot fi prelucrate pe minut de către sistemul de baze de date
  - Utilizarea spațiului de memorare: dimensiunea spațiului pe disc utilizat de fişierele bazei de date şi de structurile de acces la date

Implementarea și testarea bazelor de date

- **Implementarea unei baze de date relaționale** înseamnă:
  - **Creearea obiectelor bazei de date** (tabele, vederi, indexuri), pe baza proiectului logic şi a proiectului fizic, folosind limbajul de descriere a datelor (LDD) oferit de sistemul SGBD ales, sau toolset-uri grafice (de ex. *table editor*) sau prin executia scriptului generat de un toolset de proiectare
  - **Implementarea procedurilor pentru tratarea constrângerilor explicite** (asertiuni, dependențe de date care nu sunt determinate de chei ale relațiilor), a căror previziune şi documentare a fost realizată în faza de proiectare logică a bazei de date
- **Testarea funcționării bazelor de date:**
  - **Popularea bazei de date** cu date obținute prin conversia unor date existente sub formă de fişiere sau introduse direct în tabele
  - **Monitorizări**, întreținere, teste de performanță
- **Modalități de proiectare și implementare a bazelor de date:**
  - **Proiectarea directă (normală - *forward engineering*)**: se pornește de la cerințe, apoi diagrama E/A, schema logică, implementarea schemei (crearea relațiilor, vederilor etc.); în MySQL Workbench cu comanda *Forward Engineering se obține implementarea bazei de date pornind de la schema logică*
  - **Proiectarea inversă (*reverse engineering*)**: se pleacă de la baza de date și se obține schema logică; în MySQL Workbench - comanda *Reverse Engineering*

Dezvoltarea aplicațiilor de baze de date

- O aplicație de baze de date constă din:
  - Programe care se execută în SGBD (proceduri stocate, funcții, triggere, cursoare)
  - Programe de baze de date care se execută în afara SGBD; acestea asigură:
    - Interfața (grafică) cu utilizatorii
    - Algoritmi de calcul (business logic)
    - Interfața cu SGBD



- Programele din SGBD (proceduri stocate, funcții definite de utilizator, trigere):
  - Limbajul SQL2, folosit in SGBD-urile relaționale este un limbaj neprocedural de aceea SGBD-urile mai folosesc si extensii procedurale a limbajului SQL2:
    - PL/SQL in sistemele Oracle, PL/PLGSQL in sistemele PostGreSQL
    - Transact-SQL in sistemele Microsoft SQL Server
    - Extensie SQL in MySQL (asemanatoare cu Transact SQL) etc.
- Programele de baze de date folosesc mai multe limbaje și biblioteci (interfețe):
  - Limbajul SQL integrat intr-un limbaj de nivel inalt (Embeded SQL)
  - Interfete de programare a aplicatiilor (API) (*call level interface*)

Limbaje procedurale de extensie a SQL

- Extensiile procedurale ale limbajului SQL:
  - Ofera suport pentru crearea procedurilor stocate, a funcțiilor definite de utilizator, a declanşatorilor (triggere) și a cursoarelor)
  - Definesc: variabile, blocuri de executie, instructiuni pentru controlul ordinii de execuție (bucle while, for, repeat; instrucțiuni condiționale if...else; etc.), instrucțiuni SQL extinse
- *Variabilele* sunt folosite pentru stocarea în memorie a unor valori care pot fi testate sau modificate şi pot fi folosite pt transferul datelor către şi de la tabele
- *Variabilele locale* au ca domeniu de definiție blocul, procedura, functia sau trigger-ul în care au fost declarate
- Un bloc este un grup de instructiuni delimitat prin instructiunile: BEGIN ... END; un bloc este considerat o instructiune compusă
- O variabilă locală se declară si se initializeaza diferit de la un SGBD la altul. ex:
  - in Transact SQL: DECLARE @contor INT SELECT @contor = 0
  - in PL/SQL (Oracle): DECLARE CONTOR := 1;
  - in MySQL: DECLARE contor INT; SET contor = 0;
- *Ordinea de execuție a instrucțiunilor* este controlată prin instrucțiuni ca:

  BEGIN...END REPEAT...UNTIL FOR
  GOTO WHILE
  IF...ELSE BREAK
  RETURN CONTINUE

Instructiuni SQL extinse

- Extensiile procedurale definesc clauze suplimentare in instructiunile SQL, astfel încât acestea să poată fi folosite în combinație cu variabilele locale
- De exemplu - instrucțiunea SELECT prin care se încarcă valori ale unor atribute selectate din baza de date in variabile locale:
  - In Transact-SQL: SELECT @var1 = col1, @var2 = col2, ... @varn = coln
    FROM lista_tabele WHERE conditie
  - In PL/SQL (Oracle): SELECT lista_coloane **INTO** lista_variabile
    FROM lista_tabele [WHERE conditie] [optiuni]
  - In MySQL: SELECT lista_coloane **INTO** lista_variabile
    FROM lista_tabele [WHERE conditie] [optiuni]
- Astfel de instrucțiuni sunt utile pentru interogările care returnează o singură linie; de exemplu (mySQL)

```
DECLARE s_nume, s_prenume varchar(20);
SELECT Nume, Prenume INTO s_nume, s_prenume
    FROM ANGAJATI WHERE IdAngajat = 5;
```

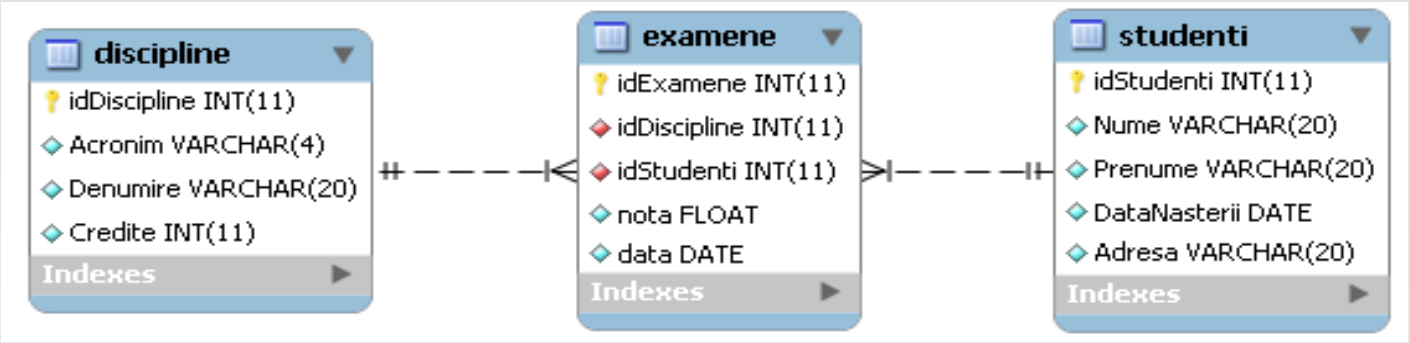
- Dacă interogarea returnează mai multe linii, variabilele locale sunt setate cu valorile atributelor din prima linie a rezultatului, iar celelalte linii se pierd; în acest caz se foloseste un cursor
- Variabilele locale pot fi folosite și in instructiunile INSERT sau UPDATE (ca valori introduse) și în clauza WHERE pentru orice instrucțiune SQL

Proceduri stocate

- **O procedură stocată** (*stored procedure*) este o procedură care implementează o parte din algoritmii de calcul ai aplicațiilor
- O procedură stocată se definește, se compilează și se memorează in baza de date, apoi poate fi apelată de oricâte ori cu instrucțiunea call
- Procedurile stocate se definesc folosind extensiile procedurale ale SQL:
  - In Transact-SQL: **CREATE PROCEDURE** nume_proc [parametri] AS instruct_compusa
  - In PL/SQL: **CREATE PROCEDURE** nume_proc [parametri] AS instruct_compusa
  - In MySQL: **CREATE PROCEDURE** nume_proc [parametri] instruct_compusa
- Parametrii pot fi de intrare (IN), de iesire (OUT) sau de intrare-iesire (INOUT); apelul unei proceduri stocate de către un client (aplicație) produce execuția de către SGBD a tuturor instrucțiunilor procedurii şi returnarea rezultatelor în parametrii OUT si INOUT
- Avantaje - imbunatatirea performantelor sistemului prin:
  - Scaderea comunicației între aplicație şi serverul bazei de date
  - Scaderea timpului de execuție a sarcinii respective, dat fiind că procedura stocată este deja compilată, optimizată si memorata, putând fi apelată oricând, de oricâți clienți
- Dezavantaje: congestionarea serverului si scaderea performantelor acestuia, dacă prea multe aplicații execută operațiile de prelucrare pe server prin intermediul procedurilor stocate

Exemplu: Procedura stocata in MySQL

- Folosim tabelele: Studenti, Examene, Discipline:



- Procedura stocata: Calculul mediei notelor la o disciplina data:

```
DELIMITER $$  /* se redefineste delim. deoarece ;  */
              /* este obligatoriu intre instr. din blocuri */
DROP PROCEDURE IF EXISTS SP_Media $$
CREATE PROCEDURE SP_Media (OUT media float, IN acro varchar(4))
BEGIN
   SELECT AVG(nota) INTO media FROM DISCIPLINE, EXAMENE
      WHERE DISCIPLINE.idDiscipline = EXAMENE.idDiscipline
      AND Acronim = acro;
END $$
DELIMITER ;
```

- Apelul procedurii:

```
call SP_Media(@media,'PBD');/* o var @ este implicit locala */
select @media;
```

Functii definite de utilizator

- ***O funcție definită de utilizator*** (*user-defined function*) este o funcție memorată in baza de date, la fel ca o procedură stocată
- O funcție are numai parametri de intrare, returnează întotdeauna o valoare şi poate fi folosită direct în expresii (o procedură stocată poate să returneze zero, una sau mai multe valori prin parametrii de tip OUT sau INOUT)
- Functiile se definesc folosind extensiile procedurale ale limbajului SQL:
  In Transact-SQL: **CREATE FUNCTION** nume_func [parametri] AS instruct_compusa
  In PL/SQL: **CREATE FUNCTION** nume_func [parametri] AS instruct_compusa
  In MySQL: **CREATE FUNCTION** nume_func [parametri] instruct_compusa
- Exemplu de creare a unei functii in MySQL:

```
DELIMITER $$
DROP FUNCTION IF EXISTS Func_Media $$
CREATE FUNCTION Func_Media(acro varchar(4)) RETURNS float
BEGIN
   DECLARE media float;
   SELECT AVG(nota) INTO media FROM DISCIPLINE, EXAMENE
        WHERE DISCIPLINE.idDiscipline = EXAMENE.idDiscipline
        AND Acronim = acro;
   RETURN media;
END $$
DELIMITER ;
```

- Utilizarea valorii returnate de o functie: `select Func_Media('PBD');`

Cursoare

- ***Un cursor*** (*cursor*) este o structură (un buffer) care permite memorarea unei mulțimi de linii returnate de o instrucțiune de interogare, urmata de extragerea si prelucrarea (eventual repetată) in programele de aplicatii a fiecarei linii
- Cursoarele se pot crea folosind limbajul SQL și extensiile procedurale ale acestuia sau biblioteci de conectare la SGBD-uri
- Instrucțiunile SQL de definire şi de operare a cursoarelor:
  - Definire cursor:

    DECLARE nume_cursor [OPTIUNI] CURSOR **FOR** instructiune_select;
  - Deschidere cursor: se execută instrucțiunea SELECT și se încarcă datele în cursor:

    OPEN nume_cursor;
  - Extragerea unei (sau mai multor) linii dintr-un cursor de la pozitia curenta:

    FETCH [FROM] nume_cursor **INTO** lista_variabile;
  - Inchiderea cursor: **CLOSE** nume_cursor;
- Cursoarele (mulțimea de linii rezultate) pot fi memorate:
  - la server, iar clientul primeşte câte o linie (sau un grup de linii) de la server la fiecare instrucțiune de extragere FETCH
  - la client şi liniile sunt folosite direct în programul respectiv
- In general, cursoarele la server sunt mai avantajoase decât cursoarele la client, deoarece cursoarele la client necesită ca toate liniile rezultat să fie transferate dintr-o dată de la server la client

Exemplu: Cursor într-o procedură stocată MySQL (1)

- Procedura: Calculul mediei notelor unui student dat (nume, prenume)

```
DELIMITER $$
DROP PROCEDURE IF EXISTS Medii_Studenti $$
CREATE PROCEDURE Medii_Studenti(OUT media float, IN s_nume
  varchar(20), IN s_prenume varchar(20))
BEGIN
  DECLARE done INT DEFAULT 0; DECLARE student, id_student INT;
/* Creare cursor */
DECLARE cursor_examene CURSOR FOR
   SELECT idStudenti, avg(nota) FROM EXAMENE
        GROUP BY idStudenti;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
  /*Aflare id student dat cu nume, prenume in var. student */
SELECT idStudenti INTO student FROM STUDENTI
        WHERE Nume = s_nume  AND Prenume = s_prenume;
  /* Deschidere cursor */
OPEN cursor_examene;
  /* Parcurgere linii cursor */
REPEAT  FETCH cursor_examene INTO id_student, media;
UNTIL done = 1 OR id_student = student
END REPEAT;
CLOSE cursor_examene;
END$$
DELIMITER ;
```

Exemplu: Cursor intr-o procedura stocata MySQL (2)

- Pentru parcurgerea liniilor cursorului se defineste un handler pentru conditia de terminare a parcurgerii liniilor cursorului (not found); un handler este un fel de rutina de tratare a exceptiilor
- Apelul procedurii:

```
call Medii_Studenti(@media, 'Popescu', 'Marius');
select @media;
```

- Se obtine rezultatul: `@media`

(valoare ce depinde de continutul tabelului)

- Parcurgerea liniilor cursorului se poate face si cu instructiunea while:

```
FETCH cursor_examene INTO id_student, media;
WHILE done = 0 AND id_student <> student DO
 FETCH cursor_examene INTO id_student, media;
END WHILE;
```

- Declararea unei variabile locale (cu instructiunea DECLARE) se poate face numai intr-un bloc BEGIN ... END si numai la inceputul acestuia
- Declaratiile trebuie sa fie facute intr-o anumită ordine: variabile locale, cursoare, handlere

Triggere

- ***Un trigger*** este o procedură stocată specială, care este executată automat atunci când se efectuează operații de actualizare a relațiilor (INSERT, DELETE, UPDATE)
- Triggerele pot fi creeate folosind extensiile procedurale ale limbajului SQL; sintaxa difera de la un SGBD la altul (sunt neportabile):

  In Transact-SQL: **CREATE TRIGGER** nume_trigger ON tabel

  {FOR|AFTER|INSTEAD OF} {[DELETE][,INSERT][,UPDATE]} AS instruct_comp.

  In PI/SQL (Oracle): **CREATE TRIGGER** nume_trigger {BEFORE|AFTER} [INSERT,

  DELETE, UPDATE] [FOR EACH ROW [WHEN conditie]] CALL procedura

  In MySQL: **CREATE TRIGGER** nume_trigger {BEFORE|AFTER} [INSERT, DELETE, UPDATE] ON tabel FOR EACH ROW instructiune_comp.
- Utilizarea triggerelor:
  - Generarea automată a unor valori care rezultă din valori ale altor atribute (exemplul care urmează)
  - Jurnalizarea transparentă a evenimentelor sau culegerea de date statistice în legătură cu accesarea relațiilor
  - Impunerea constrângerile explicite cum sunt dependențele de date (dependențe funcționale, multivalorice sau de joncțiune) care nu sunt determinate de chei, pentru menținerea integrității bazei de date

Exemplu: trigger MySQL

- Se defineste un trigger care generează coloana ‘nota’ in tabelul examene_2 (idExamene, idDiscipline, notaLab, notaExam, nota):

```
DELIMITER $$
DROP TRIGGER IF EXISTS calcul_nota $$
CREATE  TRIGGER calcul_nota BEFORE UPDATE ON `examene_2`
    FOR EACH ROW
    BEGIN
        SET NEW.nota = NEW.notaLab + NEW.notaExam;
    END  $$
DELIMITER ;
```

- Instructiunile după FOR EACH ROW se executa de fiecare dată când triggerul este activat, ceea ce se întamplă la fiecare linie afectata de instructiunea de declansare a triggerului (UPDATE in exemplul dat):

```
update examene_2 set notaLab = 2 , notaExam = 5
            where idStudenti=1 AND idDiscipline = 2;
```

- OLD si NEW sunt tabele cu aceeași schemă ca și tabelul pe care este definit trigger-ul, dar cu o singură linie, cea afectată de trigger:
  - pentru triggere INSERT se poate folosi numai NEW
  - pentru triggere DELETE se poate folosi numai OLD
  - pentru triggere UPDATE se poate folosi OLD (pentru valorile dinainte de UPDATE) sau NEW (pentru valorile actualizate).

Comunicația aplicațiilor cu SGBD

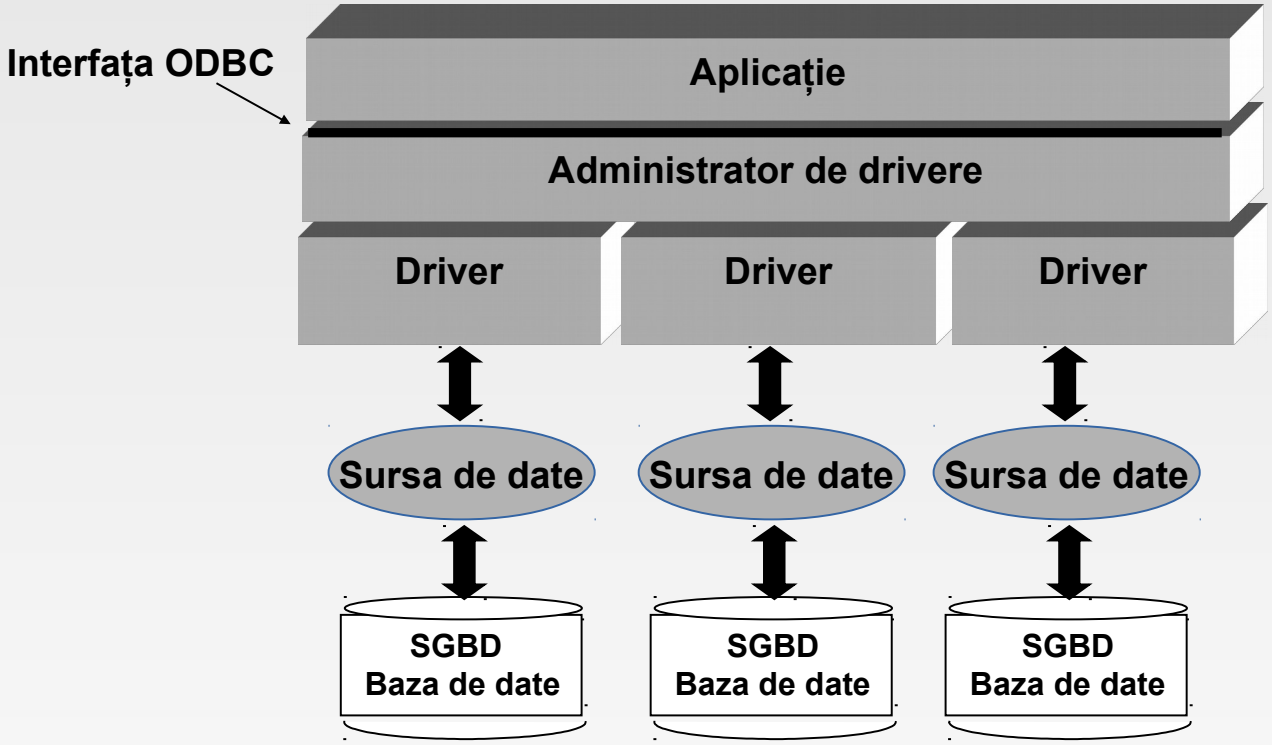
- Comunicația aplicațiilor cu SGBD se realizează prin transmiterea de instrucțiuni SQL către SGBD, folosind:
  - Limbaje SQL integrate (Embeded SQL)
  - Interfețe și biblioteci de conectare la SGBD
- **Într-un limbaj SQL integrat** (Embeded SQL) instrucțiunile SQL sunt incluse direct în codul programului sursă scris într-un limbaj gazdă de nivel înalt
- Standardele SQL definesc suport integrat pentru limbajele PL/1, C, Pascal, Cobol, Fortran, Java
  - Microsoft SQL Server: limbajul SQL este integrat în limbajul gazdă C sub numele de ESQL/C (Embedded SQL for C)
  - Oracle: limbajul SQL este integrat în limbajul Java, sub numele de SQLJ
  - MySQL: limbajul SQL este integrat în limbajul C prin biblioteca mysqld
- Instrucțiunile SQL integrate în programul în limbajul gazdă sunt prelucrate de un preprocesor și transformate în apeluri de funcții ale unei biblioteci a SGBD-ului
- Rezultatul preprocesării este un program sursă în limbajul gazdă, care poate fi compilat cu compilatorul limbajului gazdă respectiv şi apoi legat (link) cu bibliotecile de sistem şi cu biblioteca SGBD-ului
- Programul sursă rezultat al preprocesării conține un amestec de instrucțiuni în limbajul gazdă cu apeluri de funcții ale bibliotecii SGBD-ului și este foarte greu de depanat
- De aceea limbajul SQL integrat este rareori folosit

Interfețe si biblioteci de conectare la SGBD

- Există două categorii de interfete de conectare a aplicatiilor la SGBD:
  - Interfete specifice unui anumit SGBD
  - Interfete independente de SGBD
- ***Interfetele specifice unui anumit SGBD*** sunt definite prin biblioteci care contin funcții şi macrodefiniții ce permit aplicațiilor client să interacționeze cu SGBD-ul pentru conectare la baza de date, execuție instrucțiuni SQL, etc.
- Astfel de interfețe sunt specifice fiecărui SGBD și nu oferă portabilitate de la un SGBD la altul; de exemplu:
  - Biblioteca C pentru sistemul Microsoft SQL Server - DB-Library for C
  - Biblioteca MySQL C API
- ***Interfețe independente de SGBD***, cu un grad ridicat de portabilitate, care pot fi folosite pentru mai multe tipuri de SGBD-uri; cele mai cunoscute sunt:
  - Interfata ODBC (Open DataBase Connectivity)
  - Interfata JDBC (Java DataBase Connectivity)

Interfata ODBC

- Tehnologia ODBC (Open Database Connectivity) - interfață de programare a aplicațiilor prin apel de funcții (în limbajul C) independente de SGBD
- Independența se obține prin ***drivere*** specifice fiecărui SGBD și ***surse de date*** care asociază fiecare bază de date dintr-un SGBD cu un driver din bibliotecă
- Pentru conectarea la o bază de date, se specifică o anumită sursă de date, iar administratorul de drivere direcționează apelurile de funcții din aplicație către driverul cu care este asociată sursa de date respectivă



Interfața JDBC (1)

- JDBC este o interfață de programare a aplicațiilor de baze de date independentă de platformă şi de SGBD, asemănătoare cu ODBC (dar este în limbajul Java)
- La fel ca și ODBC, interfața JDBC constă din mai multe niveluri: administrator de drivere, drivere, surse de date
- Administratorul de drivere JDBC este reprezentat prin clasa **`DriverManager`** care asigură interacțiunea cu programul de aplicație, pe de o parte, şi selectarea driverului instalat pentru un anumit SGBD, pe de altă parte
- Încărcarea driverelor se poate face prin apelul metodei statice **`Class.forName (nume_driver)`**, care încarcă în mod explicit clasa driverului dată ca argument
- Într-un program JDBC, o conexiune la o bază de date se definește printr-un obiect instanță a unei clase care implementează interfața **`Connection`**, care se obține prin apelul funcției statice **`getConnection()`** a clasei **`DriverManager`**
- Acestei funcții i se pasează ca argument un şir de caractere care reprezintă adresa bazei de date într-un format URL (Uniform Resource Locator), și alte informații care depind de driver (de ex. pentru driverul MySQL, se transmite numele utilizatorului MySQL și parola)
- Exemplu – codul de creeare a unei conexiuni la o bază de date mysql:

```
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(
      "jdbc:mysql://hostname:port/dbname","username", "password");
```

Interfața JDBC (2)

- Interfața JDBC oferă posibilitatea de construire a instrucțiunilor SQL, care sunt transmise SGBD-ului pentru a fi executate
- În interfața JDBC termenul de "instrucțiune" (*statement*) are urm. semnificații:
  - Instrucțiune de program (Java sau alt limbaj)
  - Obiect instrucțiune JDBC (**`Statement`**), creat pentru comunicația cu SGBD-ul
  - Instrucțiunea SQL, construită ca un şir de caractere care este transmis SGBD-ului ca argument al unei metode a unui obiect instrucțiune JDBC (**`executeQuery()`** etc.)
- La fel, termenul "interfață" (*interface*) se poate referi la:
  - Tip de date (interfață Java)
  - Mulțimea metodelor publice ale unei clase (interfață C++)
  - Bibliotecă pentru com. între module de program (API – *Application Program Interface*)
- Pentru construirea instrucțiunilor SQL se folosesc metode ale interfeței **`Connection`**, care creează un obiect instrucțiune JDBC şi returnează o referință la interfața pe care clasa obiectului o implementează:
  - **`Statement`** – este interfața folosită pentru transmiterea instrucțiunilor SQL simple, fără parametri; funcția **`createStatement()`** a interfeței **`Connection`** returnează un obiect instrucțiune JDBC indicat printr-o referință de tip **`Statement`**
  - **`PreparedStatement`** – este o interfață folosită pentru transmiterea instrucț SQL precompilate (pregătite) şi poate primi unul sau mai mulți parametri de intrare (IN); funcția **`prepareStatement()`** a clasei **`Connection`** returnează un obiect instrucțiune JDBC indicat printr-**o** referință de tip **`PreparedStatement`**
  - **`CallableStatement`** – este o interfață folosită pentru apelul procedurilor stocate şi poate manevra atât parametri de intrare (IN), cât şi parametri de ieşire (OUT) şi parametri de intrare-ieşire (INOUT); funcția **`prepareCall()`** a interfeței **`Connection`** returnează un obiect instr. JDBC indicat prinr-o referință de tip **`CallableStatement`**

Interfața JDBC (3)

- Interfața **`Statement`** prevede trei metode de execuție a instrucțiunilor SQL: **`executeQuery()`, `executeUpdate()`** şi **`execute()`**
  - Metoda **`executeQuery()`** se foloseşte pentru execuția instr SQL care returnează ca rezultat o singură mulțime de linii (result set), aşa cum este instrucțiunea **`SELECT`**
  - Metoda **`executeUpdate()`** se foloseşte pentru instrucțiunile SQL de definire a datelor (**`CREATE TABLE`, `DROP TABLE`**, etc.) şi pentru instrucțiunile de actualizare a tabelelor (**`INSERT`, `DELETE`, `UPDATE`**) care returnează un contor de actualizare (un întreg) ce reprezintă numărul de linii afectate
  - Metoda **`execute()`** se foloseşte pentru instrucțiunile SQL care returnează mai mult de o mulțime de linii rezultat sau mai mult de un contor de actualizare
- Un program JDBC de conectare la o bază de date şi de execuție a unor instrucțiuni SQL simple este prezentat în exemplul următor, dezvoltat în Eclipse; programul afișează lista studenților din tabelul *studenti* din baza de date *facultate*
- Pentru încărcarea driverului JDBC pentru MySQL, se include în proiect arhiva mysql-connector-java-5.0.8.bin.jar
- După încărcarea driverului JDBC şi creearea conexiunii la baza de date, se creează o instrucțiune simplă (fără parametri) de tip **`Statement`**.
  - Pentru execuția unei instrucțiuni SQL de tip **`UPDATE`** (sau **`INSERT`, `DELETE`**) se apează funcția **`executeUpdate()`** a obiectului de tip **`Statement`**
  - Pentru o interogare (instr. SQL **`SELECT`**) se apelează funcția **`executeQuery()`**, care returnează un obiect indicat printr-o referință la interfața **`ResultSet;`**
  - Acest obiect conține mulțimea de linii rezultat, care pot fi parcurse și afișate (sau executate alte operații)

Exemplu de program JDBC

```
package facultate;
import java.sql.*;
public class Lista_Studenti{
  public static void main (String[] args) {
    String dbUrl = "jdbc:mysql://localhost:3306/facultate";
    String user = "root", passw = "parola";
    try {
      //Incarcarea driverului si crearea conexiunii
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.getConnection(dbUrl,user,passw);

      // Creearea instructiunii JDBC (stmt) si executie instr SQL
      Statement stmt = conn.createStatement();
      stmt.executeUpdate("UPDATE studenti SET Adresa = 'Buzau'
                                    WHERE idStudenti=1");
      ResultSet rs = stmt.executeQuery ("SELECT idStudenti, Nume,
                                    Prenume, Adresa FROM studenti");
      System.out.println("Lista Studenti");
      // Afisarea liniilor rezultatului
      while(rs.next()) System.out.println(rs.getInt(1) + " " +
          rs.getString(2)+" "+ rs.getString(3) +" "+ rs.getString(4));
      // Inchiderea instructiunilor si a conexiunii
      stmt.close();
      conn.close();
    } catch(Exception e) { e.printStackTrace(); }
  }
}
```

Interfața ResultSet și tipurile de date JDBC

- ***Interfața ResultSet***. În interfața JDBC nu sunt prevăzute posibilități de creare explicită a cursoarelor, dar un obiect indicat printr-o referință de tip **`ResultSet`**, returnat de o instrucțiune de interogare reprezintă, de fapt, *un cursor implicit la client*, care gestionează mulțimea de linii rezultat al interogării
- ***Tipurile de date JDBC***. Corespondența dintre tipurile de date SQL şi tipurile de date Java nu este foarte simplu de realizat, datorită faptului că diferitele sisteme SGBD implementează dialecte diferite ale limbajului SQL
- Pentru a asigura portabilitatea programelor JDBC (independența față de SGBD), în interfața JDBC sunt definite tipuri SQL generice care corespund în general standardului SQL2, iar driverul respectiv face conversia tipului JDBC în tipul SQL corespunzător, suportat de SGBD-ul respectiv
- Tipurile de date JDBC sunt definite ca și constante **`java.sql.Types.CHAR`**, etc:
  - **`CHAR, VARCHAR`** – corespund diferitelor variante de tipuri SQL **`CHAR`** şi **`VARCHAR`**; în Java, aceste tipuri corespund clasei **`String`**
  - **`INTEGER, SMALLINT`** – corespund tipurilor SQL **`INTEGER`**, respectiv **`SMALLINT`**; în Java aceste tipuri corespund tipurilor **`int`**, respectiv **`short`**
  - **`FLOAT, DOUBLE`** – corespund tipurilor SQL **`FLOAT`**, respectiv **`DOUBLE`**; în Java aceste tipuri corespund tipurilor **`float`**, respectiv **`double`**
  - **`NUMERIC, DECIMAL`** – corespund tipurilor SQL **`NUMERIC`**, **`DECIMAL`**, care sunt foarte asemănătoare între ele; în Java, acestor tipuri le corespunde fie clasa **`java.math.BigDecimal`**, care le memorează ca numere zecimale cu precizia dorită şi permite operații asupra lor, fie clasa **`String`**, care le memorează ca şiruri de caractere.

JDBC – Instrucțiuni SQL dinamice

- O instrucțiune SQL dinamică (sau parametrizată) conține valori care se furnizează în cursul execuției, putând fi necunoscute la compilare
- În astfel de situații, se creează un obiect instrucț cu interfață **`PreparedStatement`**, folosind metoda **`prepareStatement()`** a interfeței **`Connection`**
- Această metodă primeşte ca argument şirul de caractere care reprezintă instrucțiunea SQL, în textul căreia se introduce câte un marcaj de parametru (semnul întrebării) pentru fiecare valoare care se va furniza în cursul execuției
- Parametrii instrucțiunii sunt identificați pozițional, începând cu 1
- Asocierea (legătura - binding) între fiecare parametru marcat (care este un atribut - coloană a unui tabel) şi o variabilă din programul de aplicație, se face folosind o metodă de tipul **`set`** a interfeței **`PreparedStatement`** (**`setInt(), setString()`**, etc.); aceste metode asigură şi conversia corespunzătoare a datelor
- O instrucțiune de tipul **`PreparedStatement`** poate fi executată cu una din metodele **`executeQuery(), executeUpdate()`** sau **`execute()`**, redefinite în interfața **`PreparedStatement`** ca metode fără nici-un argument, dat fiind că instrucțiunea SQL pe care o execută a fost deja pregătită
- La execuția unei astfel de metode (**`executeQuery()`** etc.) se încarcă mai întâi valoarea curentă a variabilelor locale în parametrii instrucțiunii SQL, conform asocierii (binding) definite, apoi instrucțiunea SQL este transmisă SGBD-ului
- În aplicația Java din exemplul următor este definită funcția **`Medii_Studenti()`** care calculează media notelor unui student dat prin nume și prenume; această funcție poate înlocui procedura stocată **`Medii_Studenti`** din baza de date, descrisă înainte

Exemplu JDBC – Instrucțiuni SQL dinamice (1)

```
package facultate;
import java.sql.*;
public class Calcul_Medii_Studenti{
  public static void main (String[] args) {
    String dbUrl = "jdbc:mysql://localhost:3306/facultate";
    String user = "root", passw = "parola";
    Connection conn = null;

    try {Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.getConnection(dbUrl,user,passw);
    } catch (Exception e){ printStackTrace();}

    float media = Medii_Studenti(conn, args[0], args[1]);
    System.out.println("Calcul medii studenti: "
                       + args[0] + " " + args[1] + ": " + media);
    try { conn.close();
    } catch (SQLException e){e.printStackTrace();}
  }
  public static float Medii_Studenti (Connection conn,
      String l_nume, String l_prenume) {
      ..............................
  }
}
```

- Parametrii funcției de calcul `Medii_Studenti()` se dau ca argumente de execuție ale metodei `main()` a programului (args[0], args[1])
- În Eclipse, se configurează lansarea programului (cu comanda Configure Run), introducând valorile dorite pentru args[0] și args[1]

Exemplu JDBC – Instrucțiuni SQL dinamice (2)

```
public static float Medii_Studenti (Connection conn,
          String l_nume, String l_prenume) {
  float media = 0.0f;
  try {
    PreparedStatement pstmt = conn.prepareStatement(
     "SELECT idStudenti FROM studenti WHERE Nume=? AND Prenume=?");
    // Binding - asocierea coloanelor rezultatului cu var. locale
    pstmt.setString(1, l_nume);
    pstmt.setString(2, l_prenume);
    // Execuție instrucțiune JDBC
    ResultSet rs = pstmt.executeQuery();
    rs.next();
    int id_student = rs.getInt(1);
    // Mediile tuturor studentilor
    Statement stmt = conn.createStatement();
    rs = stmt.executeQuery(
     "SELECT idStudenti, avg(nota) FROM examene GROUP BY idStudenti;");
    // rs - mulțime de linii cu coloanele: (1)idStudenti,(2) avg(nota)
    while(rs.next()) {
      media = rs.getFloat(2);
      if (rs.getInt(1) == id_student) break;
    }
    // Inchiderea instructiunilor JDBC
    stmt.close();
    pstmt.close();
  } catch(Exception e) { e.printStackTrace(); }
  return media;
}
```

JDBC – Apelul procedurilor stocate

- Pentru apelul procedurilor stocate se creează o instrucțiune JDBC de tipul **`CallableStatement`** prin apelul metodei **`prepareCall()`** a interfeței **`Connection`**
- Argumentul acestei metode este instrucțiunea SQL prin care se apelează procedura respectivă, în care o parte din param de intrare (IN), param de ieșire (OUT) sau de intrare/ieșire (INOUT) se pot marca prin semnul întrebării ca variabile necunoscute:

```
CallableStatement cstmt = conn.prepareCall("call Medii_Studenti(?,?,?)");
```

- Parametrii procedurii sunt identificați pozițional, începând cu 1
- Înainte de execuția instrucțiunii, se asociază parametrii de intrare cu variabile locale din program, iar pentru parametrii de ieșire se înregistrează tipul JDBC returnat XXX (care poate fi INTEGER, FLOAT, VARCHAR etc.)
- La apelul metodei **`execute()`** a instrucțiunii **`CallableStatement`**, se atribuie parametrilor de intrare ai procedurii valorile variabilelor cu care aceștia sunt asociați și apoi se transmite instrucțiunea SQL call către SGBD
- Valoarea returnată se recuperează cu o metoda **`getXXX()`** a instrucțiunii **`CallableStatement`**, care face și conversia de tip de date necesară
- Se observă că aceeași operație de calcul (în exemplu, calculul mediei unui student dat prin nume și prenume) se poate defini:
  - Fie ca procedură stocată în baza de date, care poate fi apelată din programele de aplicații
  - Fie ca funcție în programul de aplicație
- Care soluție este mai bună, depinde de caracteristicile sistemului de baze de date (puterea de calcul a serverului SGBD, nr de aplicații client care se conectează etc.)

Exemplu JDBC - Apelul procedurilor stocate

```
package facultate;
import java.sql.*;
public class Call_Medii_Studenti{
  public static void main (String[] args) {
    String dbUrl = "jdbc:mysql://localhost:3306/facultate";
    String user = "root", passw = "parola";
    try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.getConnection(dbUrl,user,passw);

      // Creeare instructiune JDBC CallableStatement
      String sql = "call Medii_Studenti(?,?,?)";
      CallableStatement cstmt = conn.prepareCall(sql);
      //Asociere parametri IN cu valorile argumentelor
      cstmt.setString(2, args[0]);
      cstmt.setString(3, args[1]);
      // Inregistrare tip parametru de iesire
      cstmt.registerOutParameter(1, java.sql.Types.FLOAT);
      // Executie - apel procedura stocata
      cstmt.execute();
      //Extragere valoare parametru OUT
      float media = cstmt.getFloat(1);
      System.out.println("Call procedura stocata: " +
                         args[0] + " " + args[1] + ": " + media);
      // Inchiderea instructiunii si a conexiunii
      cstmt.close();
      conn.close();
    } catch(Exception e) { e.printStackTrace(); }
  }
}
```