Cap.4. Dezvoltarea SBD

CUPRINS

1. Etapele de dezvoltare a SBD

DEFINITII

Sistemul informatic (information system) al unei organizatii: include toate resursele acelei organizaţii care sunt implicate **în colectarea, administrarea, utilizarea şi diseminarea informaţiilor.**

Istoric sistemele informatice:

- **pana in 1970**: sisteme de fişiere (pe disc sau bandă magnetică)
- **in prezent**: SBD, care permit gestionarea unor volume de date mari într-un timp redus, asigurand protecţia si securitatea datelor

1. Etapele de dezvoltare a SBD

DEFINITII

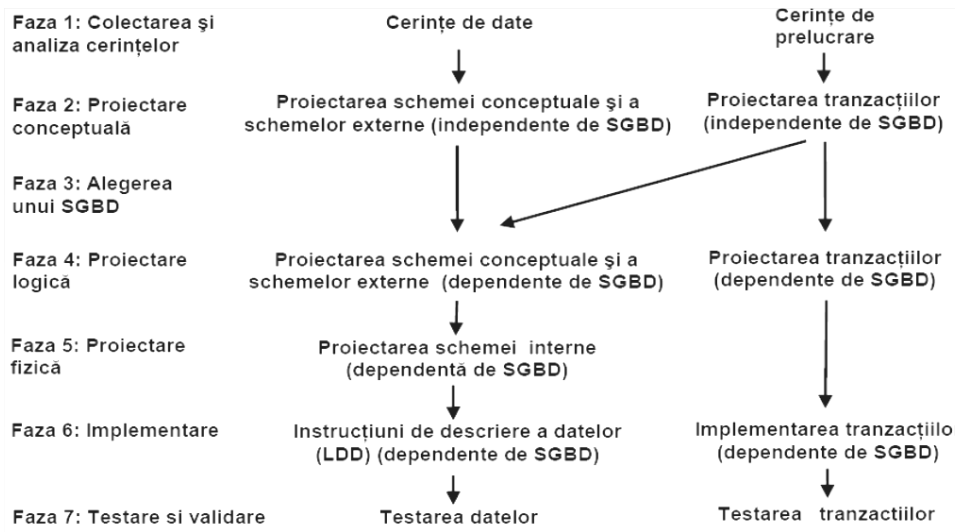
Etapele de dezvoltare a sistemelor de baze de date(SBD)

1. **Analiza şi definirea SBD**: definirea scopului sistemului de baze de date, a utilizatorilor şi a aplicaţiilor acestuia
2. **Proiectarea SBD**: în această etapă se realizează proiectul logic şi proiectul fizic al sistemului, pentru un anumit SGBD ales
3. **Implementarea SBD**: este etapa în care se scriu definiţiile obiectelor bazei de date (tabele, vederi, etc.) şi se implementează aplicaţiile software
4. **Testarea şi validarea SBD**: testarea şi validarea cât mai riguroasa

Simplificat:

- ❑ **dezvoltarea structurii si a continutului** BD
- ❑ **dezvoltarea aplicatiilor** de prelucrare a datelor

2. Proiectarea SBD



FAZA 1: Colectarea si analiza cerintelor

Informatii initiale ce trebuie colectate :

- ce rezultate trebuie sa obtina utilizatorii ?
- ce informaţii initiale sunt disponibile ?
- ce aplicaţii trebuie realizate ? Ex: aplicaţii de gestiune a stocurilor, aplicaţii contabile, aplicaţii de urmărire a costurilor, aplicaţii de salarizare, etc.

Caracteristici FAZA 1:

- toate aceste informaţii sunt slab structurate, în general în limbaj natural
- pe baza acestora se pot construi diagrame, tabele, grafice etc., manual sau folosind diferite instrumente software de proiectare
- din aceste informatii trebuie extrase date precise de proiectare a BD si a aplicatiilor
- **fază mare consumatoare de timp**, dar este esentiala pentru proiectare

FAZA 2: Proiectarea conceptuala

Proiectarea BD consta din :

- proiectarea schemei conceptuale a BD
- proiectarea schemelor externe (view-uri ale utilizatorilor)
- proiectarea tranzactiilor

Dependenta proiectarii de SGBD-ul folosit:

- este recomandabil să se proiecteze mai întâi schema conceptuală de nivel înalt independentă de SGBD

Definirea schemei conceptuale:

- proiectare prin integrarea cerinţelor
- proiectare prin integrarea schemelor externe

Moduri de proiectare conceptuala:

- **Proiectare ascendenta (bottom-up):** se porneste de la schema conceptuala universala, care se rafineaza
- **Proiectare descendenta (up-bottom) :** se porneste de la diagrama E-A, definita pe baza tipurilor de entitati si a asocierilor dintre acestea

FAZA 2: Proiectarea conceptuala

Exemplu: INTREPRINDERE

EXEMPLU

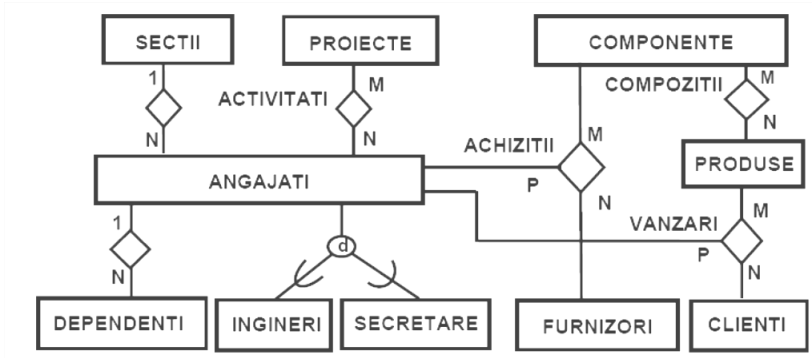
Ex.: Baza de date INTREPRINDERE cu multimile de entitati puternice:

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)

FAZA 2: Proiectarea conceptuala

Exemplu: INTREPRINDERE

Diagrama Entitate -Asociere



FAZA 3: Alegerea unui SGBD

Factori implicati in alegerea unui SGBD: tehnici, economici şi administrativi

Factorii tehnici:

- **Modelul de date** (ierarhic, reţea, relaţional, obiect-orientat, obiect-relaţional)
- **Capacitatea de stocare a datelor**
- **Posibilităţile de control al concurenţei şi de refacere a datelor**
- **Configuraţia hardware-software** necesară
- **Cerinţe de dezvoltare a programelor de aplicaţii** (limbaje, compilatoare etc.)

Factorii economici şi administrativi:

- **Costul de achiziţie pentru software:** include costul de bază + opţiuni (salvare-refacere, documentaţie, limbaje şi interfeţe de programare, drivere, etc.)
- **Costul de întreţinere**: serviciul update al versiunii SGBD.
- **Costul de pregătire a personalului:** cursurile organizate pentru persoanele care se ocupă cu întreţinerea şi operarea sistemului
- **Cunoştinţele de programare a personalului** într-un anumit SGBD

FAZA 4: Proiectarea logica

Se porneste de la **schema conceptuală** şi **schemele externe** de nivel înalt independente de SGBD (diagrama E-A), si se realizează schema logica (diagrama logica) si schemele (vederile) externe pentru sistemul SGBD ales

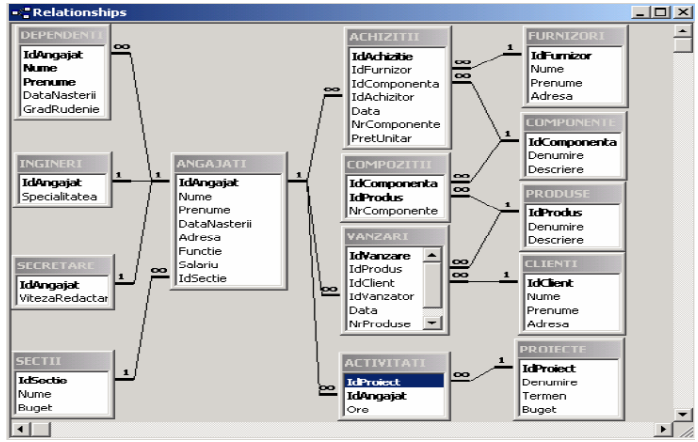
Proiectarea logică poate fi realizată în 2 subfaze:

- **transpunerea schemei logice în modelul de date al sistemului SGBD ales**, dar independent de sistemul de gestiune propriu-zis
- **rafinarea schemei logice şi a schemelor externe obţinute anterior**, a.i. să se utilizeze cât mai multe din facilităţile oferite de sistemul SGBD ales (modul de generare a cheilor primare, definirea constrângerilor, etc.)

FAZA 4: Proiectarea logica

Exemplu: INTREPRINDERE

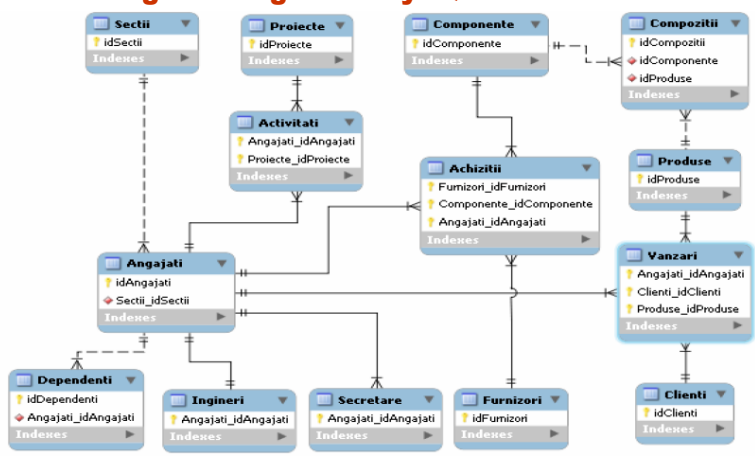
Diagrama logica in Microsoft Access



FAZA 4: Proiectarea logica

Exemplu: INTREPRINDERE

Diagrama logica in MySQL Workbench



FAZA 5: Proiectarea fizica

SGBD-urile oferă **opţiuni** de organizare a fişierelor şi a modului de acces la BD:

- indexuri
- tipuri de fisiere
- gruparea înregistrărilor corelate în aceleaşi blocuri pe disc

Proiectarea fizică a BD: alegerea acelor structuri de memorare şi de acces la fişierele BD, pentru a obţine performanţe cât mai bune pentru SGBD-ul ales, pentru cât mai multe din aplicaţiile proiectate

Parametri generali de alegere a opţiunilor :

- **Timpul de răspuns**: intervalul de timp dintre lansarea în execuţie a unei tranzacţii şi primirea răspunsului la acea tranzacţie
- **Utilizarea spaţiului de memorare**: dimensiunea spaţiului pe disc utilizat de fişierele BD şi de structurile de acces la date
- **Capacitatea tranzacţională** (transaction throughput): numărul mediu de tranzacţii care pot fi prelucrate pe minut de către SGBD

FAZA 6: Implementarea

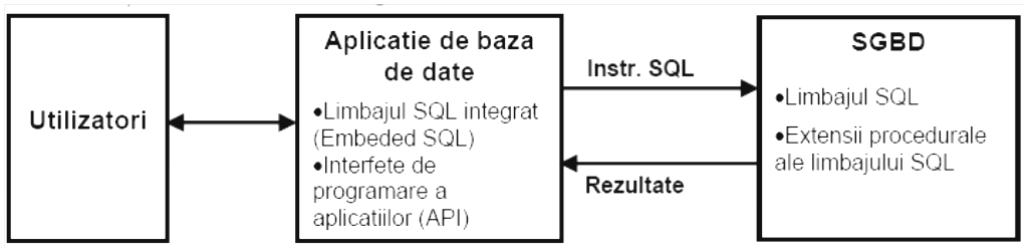
- **creearea obiectelor principale ale BD** (tabele, vederi, indexuri) : pe baza proiectului logic şi a proiectului fizic, folosind LDD oferit de sistemul SGBD ales, sau utilitare grafice (de ex. table editor)
- **implementarea procedurilor:** asigură verificarea şi impunerea constrângerilor explicite
- **popularea bazei de date:** cu informatii obţinute prin conversia unor date existente sub formă de fişiere sau introduse direct în tabele
- **implementarea programelor de aplicaţii:** utilizand limbaje procedurale de extensie SQL, limbajul SQL integrat, interfeţe şi biblioteci de programare

FAZA 7: Testarea si validarea

- testarea si validarea SBD
- monitorizarea şi întreţinerea SBD

3. Dezvoltarea aplicatiilor de BD

Aplicaţiile de BD asigura:
- ❑ interfaţa (grafică) cu utilizatorii : formulare (forms).
- ❑ implementarea algoritmilor de calcul necesari
- ❑ interfaţa cu SGBD



3. Dezvoltarea aplicatiilor de BD

Aplicaţiile de BD asigura:
- ❑ interfaţa (grafică) cu utilizatorii
- ❑ implementarea algoritmilor de calcul
- ❑ interfaţa cu SGBD

Aplicatiile de BD folosesc:

A. **Extensii ale limbajului SQL** (deoarece SQL nu are instrucţiuni de control al ordinii de execuţie a operaţiilor)

B. **Limbajul SQL integrat** intr-un limbaj de nivel inalt (Embeded SQL: C/C++, Java, Ada, PL/1, Pascal, Fortran, Cobol)

C. **Interfete de programare a aplicatiilor** (API- **Application Programming Interface** / call level interface): sunt dezvoltate ca biblioteci de functii sau de clase, iar programele de aplicatie folosesc apelul functiilor prevazute de interfata respectiva pentru a comunica cu serverul bazei de date. Exemple: pentru MySQL : C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc

A. Extensii ale limbajului SQL

Extensii ale limbajului SQL :
- 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.

Scopul utilizarii limbajelor de extensie ale limbajului SQL:
- combină instrucţiuni SQL cu **instrucţiuni pentru controlul ordinii de execuţie** (bucle while, instrucţiuni condiţionale if etc.)
- ofera **suport de crearea cursorilor**, a **procedurilor stocate**, a **funcţiilor** definite de **utilizator** şi a **triggerelor** (declanşatorilor)

CURSOR

DEFINITII

Cursor (cursor): **o structură care permite memorarea unei mulţimi de linii returnate de o instrucţiune de interogare**, urmata de extragerea si prelucrarea fiecarei linii

Moduri creare cursor: utilizand
- **limbajul SQL** sau **limbajele de extensie SQL**
- **limbajul SQL integrat** (Embedded SQL)
- **interfeţele si bibliotecile de programare a aplicaţiilor** (ODBC, JDBC)

Tipuri cursor:
- **memorat pe server**, clientul primeşte câte o linie (sau un grup de linii) de la server la fiecare instrucţiune de extragere
- **memorat direct la client**

CURSOR

Definire cursor:

```
DECLARE nume_cursor CURSOR FOR INSTRUCTIUNE_SELECT;
```

Deschidere cursor (popularea cu datele din tabele):

```
OPEN nume_cursor;
```

Extragere una (sau mai multor) linii dintr-un cursor de la pozitia curenta:

```
FETCH [FROM] nume_cursor INTO lista_variabile;
```

! Nr coloane selectate cu SELECT = nr variabile din lista_variabile FETCH

Inchidere cursor:

```
CLOSE nume_cursor;
```


EXEMPLU: Cursor MySQL

```
CREATE PROCEDURE cursdemo()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a CHAR(16);
  DECLARE b, c INT;
  DECLARE curs1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE curs2 CURSOR FOR SELECT i FROM test.t2;
  OPEN curs1;  OPEN curs2;
  read_loop: LOOP
    FETCH curs1 INTO a, b;
    FETCH curs2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
    END IF;
  END LOOP;
  CLOSE curs1;
  CLOSE curs2;
END;
```

PROCEDURI STOCATE

DEFINITII

O procedură stocată (stored procedure): **procedură care implementează** o parte din **algoritmii de calcul ai aplicaţiilor** şi care este memorată in BD

Definire procedura stocata:

- **In Transact-SQL:**

```
CREATE PROCEDURE nume_proc [parametri] AS instructiune
```

- **In PL/SQL (Oracle):**

```
CREATE PROCEDURE nume_proc [parametri] AS instructiune
```

- **In MySQL:**

```
CREATE PROCEDURE nume_proc [parametri] instructiune_compusa
```

unde parametri pot fi :de intrare (IN), de iesire (OUT) sau de intrare-iesire (INOUT);

Efect: **apelul unei proceduri stocate de către un client (aplicaţie) produce execuţia de catre SGBD a tuturor instrucţiunilor procedurii şi returnarea rezultatelor in parametri OUT si/sau INOUT**

PROCEDURI STOCATE

EXEMPLU

Ex1: Crearea unei proceduri stocate in MySQL:

```
CREATE PROCEDURE newp()
BEGIN
SELECT * FROM PERSOANE WHERE Prenume ='Petre' AND Nume='Ionescu ‘
END;
```

Ex2: Crearea unei proceduri stocate in MySQL:

```
CREATE PROCEDURE proc1(OUT param1 INT)
BEGIN
SELECT COUNT(*) INTO param1 FROM PERSOANE;
END ;
```

PROCEDURI STOCATE

DEFINITII

Avantaje: imbunatatirea performantelor sistemului prin:

- **scaderea timpului de comunicare** între aplicaţie şi serverul BD
- **scaderea timpului de execuţie** a sarcinii respective, dat fiind că procedura stocată este deja compilată, optimizată si memorata, putand fi apelata oricand, de oricati clienti

Dezavantaje:

- **congestionarea serverului** sau **scaderea performantelor serverului**, dacă prea multe aplicaţii executa operaţiile de prelucrare pe server prin intermediul procedurilor stocate

FUNCTII UTILIZATOR

DEFINITII

Funcţie utilizator (user-defined function) : o funcţie memorată in BD (ca si procedura stocată), are numai parametri de intrare, returnează întotdeauna o valoare şi poate fi folosită direct în expresii

Definire functie utilizator:

- **In Transact-SQL:**

```
CREATE FUNCTION nume_func [parametri] AS instructiune
```

- **In PL/SQL (Oracle):**

```
CREATE FUNCTION nume_func [parametri] AS instructiune
```

- **In MySQL**:

```
CREATE FUNCTION nume_func [parametri] instructiune_compusa
```

FUNCTII UTILIZATOR

EXEMPLU

Ex: Crearea unei functii utilizator de calcul a mediei studentilor in MySQL:

```
CREATE FUNCTION Func_Media(disc varchar(4)) RETURNS float
BEGIN
DECLARE nota_medie float;
SELECT AVG(nota) INTO nota_medie FROM DISCIPLINE, EXAMENE
WHERE DISCIPLINE.idDiscipline = EXAMENE.idDiscipline
AND Acronim = disc;
RETURN nota_medie;
```


TRIGGERE

DEFINITIE

Trigger: **procedură stocată specială executată automat** atunci **când se efectuează operaţii de actualizare** a relaţiilor (INSERT, DELETE, UPDATE)

Definire trigger:

- **In Transact-SQL**:

```
CREATE TRIGGER nume_trigger ON nume_tabel
{FOR|AFTER|INSTEAD OF} {[DELETE][,INSERT][,UPDATE]} AS instructiuni
```

- **In PL/SQL (Oracle):**

```
CREATE TRIGGER nume_trigger {BEFORE|AFTER} [INSERT,
DELETE, UPDATE] [FOR EACH ROW [WHEN conditie]] CALL procedura
```

- **In MySQL**:

```
CREATE TRIGGER nume_trigger ON tabel FOR EACH ROW instructiune
```

TRIGGERE

DEFINITIE

```
CREATE TRIGGER  'nume_eveniment'

BEFORE/AFTER     INSERT/UPDATE/DELETE ON BD.tabel

FOR EACH ROW

BEGIN
/*instructiuni trigger - se vor aplica fiecarei linii din tabel -
inserted/updated/deleted row */
END;
```


TRIGGERE

DEFINITIE

Parti componente	Descriere	Valori posibile
Stabilirea timpului de declansare a triggerului	Cand triggerul actioneaza fata de evenimentul declansator	BEFORE AFTER
Evenimentul declansator	Ce operatie cauzeaza activarea triggerului	INSERT UPDATE DELETE
Tipul triggerului	De cate ori se executa corpul triggerului	Statement Row
Corpul triggerului	Ce actiune executa triggerul	Executa blocul scris in PL/SQL

TRIGGERE

EXEMPLU

Ex 1: Crearea unui trigger in MySQL pentru schimbarea numelui in majuscule:

```
CREATE TRIGGER trig
AFTER INSERT ON PERSOANE
FOR EACH ROW
BEGIN
SET NEW.Nume=UPPER(NEW.Nume);
END
```


TRIGGERE

EXEMPLU

Ex 2: Crearea unui trigger in MySQL pentru majorarea pretului produselor cu 40% inainte de update:

```
CREATE TRIGGER updateProductPrice
BEFORE UPDATE ON PRODUCTS
FOR EACH ROW
BEGIN
IF NEW.prod_cost <> OLD.prod_cost THEN
SET NEW.prod_price = NEW.prod_cost * 1.40; END IF ;
END
```

Aplicatiile de BD folosesc:

A. **Extensii ale limbajului SQL** (deoarece SQL nu are instrucţiuni de control al ordinii de execuţie a operaţiilor) ☑

B. **Limbajul SQL integrat** intr-un limbaj de nivel inalt (Embeded SQL: C/C++, Java, Ada, PL/1, Pascal, Fortran, Cobol)

C. **Interfete de programare a aplicatiilor** (API- **Application Programming Interface** / call level interface): sunt dezvoltate ca biblioteci de functii sau de clase, iar programele de aplicatie folosesc apelul functiilor prevazute de interfata respectiva pentru a comunica cu serverul bazei de date. Exemple: pentru MySQL : C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc

31

B. Limbajul SQL integrat (Embeded SQL)

DEFINITIE

Limbajul SQL integrat (Embeded SQL): **instrucţiunile limbajului SQL sunt incluse direct în codul programului sursă** scris într-un limbaj gazdă de nivel înalt (C/C++, Java, Ada, PL/1, Pascal, Fortran, Cobol, etc.)

Caracteristici:

- **Controlul fluxului de operaţii este realizat prin instrucţiunile limbajului gazdă**, iar operaţiile cu BD sunt realizate prin instrucţiuni SQL
- **Instrucţiunile SQL integrate** în programul scris în limbajul gazdă sunt prelucrate si **transformate în apeluri de funcţii** ale unei biblioteci SGBD
- **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 bibliotecile SGBD-ului

B. Limbajul SQL integrat (Embeded SQL)

Exemple SQL integrate:

- **Standardul SQL2** specifică suport integrat pentru limbajele PL/1, C, Pascal, Cobol, Fortran, Mumps.
- **Oracle:** limbajul SQL a fost integrat în Java (SQLJ)
- **Microsoft SQL Server** : limbajul ESQL/C (Embedded SQL for C)
- **MySQL** : nu are suport pentru embedded SQL

Aplicatiile de BD folosesc:

A. **Extensii ale limbajului SQL** (deoarece SQL nu are instrucţiuni de control al ordinii de execuţie a operaţiilor) ☑

B. **Limbajul SQL integrat** intr-un limbaj de nivel inalt (Embeded SQL: C/C++, Java, Ada, PL/1, Pascal, Fortran, Cobol) ☑

C. **Interfete de programare a aplicatiilor** (API- **Application Programming Interface** / call level interface): sunt dezvoltate ca biblioteci de functii sau de clase, iar programele de aplicatie folosesc apelul functiilor prevazute de interfata respectiva pentru a comunica cu serverul bazei de date. Exemple: pentru MySQL : C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc

C. Interfete de programare a aplicatiilor API

DEFINITIE

Categorii de interfete de programare a aplicatiilor de BD:

- **interfete specifice** unui anumit SGBD
- **interfete independente** de SGBD

Interfete specifice unui anumit SGBD: biblioteci care contin funcţii şi macrodefiniţii ce permit aplicaţiilor să interacţioneze cu serverul BD. **Ex.:**

a) **biblioteci dezvoltate pentru limbajul C** (biblioteca C pentru Microsoft SQL Server , DB-Library for C, **MySQL C API**)

b) **biblioteci pentru alte limbaje** (C++, Perl, PHP, etc)

Interfeţe independente de SGBD: au un grad ridicat de generalitate, pot fi folosite pentru mai multe tipuri de SGBD-uri; Ex.:

c) **interfata ODBC** (Open DataBase Connectivity)

d) **interfata JDBC** (Java DataBase Connectivity)

EXEMPLU: Interfete specifice MySQL
a) biblioteci dezvoltate pentru limbajul C-C API

Table 20.4 C API Function Names and Descriptions

Function	Description
mysql_init()	Initialize global variables, and thread handler in thread-safe programs
mysql_affected_rows()	Returns the number of rows changed/deleted/inserted by the UPDATE, DELETE, or INSERT query
mysql_autocommit()	Toggles autocommit mode on/off
mysql_change_user()	Changes user and database on an open connection
mysql_character_set_name()	Return default character set name for current connection
mysql_close()	Closes a server connection
mysql_commit()	Commits the transaction
mysql_connect()	Connects to a MySQL server (this function is deprecated; us mysql_real_connect() instead)
mysql_create_db()	Creates a database (this function is deprecated; use the SQL statement CREATE DATABASE instead)
mysql_data_seek()	Seeks to an arbitrary row number in a query result set
mysql_debug()	Does a DBUG_PUSH with the given string
mysql_drop_db()	Drops a database (this function is deprecated; use the SQL statement DROP DATABASE instead)
mysql_dump_debug_info()	Makes the server write debug information to the log
mysql_eof()	Determines whether the last row of a result set has been rea

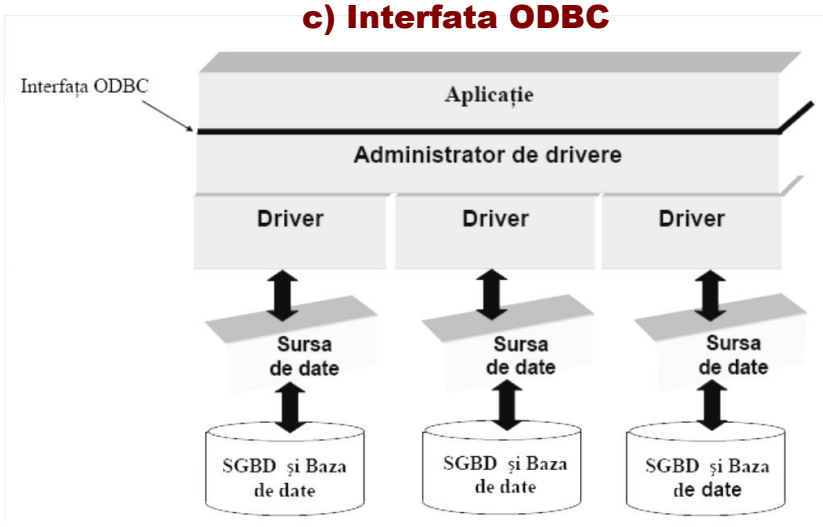
Aplicatie in C utilizand biblioteca MySQL C API

```
/* Simple C program that connects to MySQL Database server*/
#include <mysql.h>
#include <stdio.h>
main() {
   MYSQL *conn;
   MYSQL_RES *res;
   MYSQL_ROW row;
   char *server = "localhost";
   char *user = "root";
   char *password = "PASSWORD"; /* set me first */
   char *database = "mysql";
   conn = mysql_init(NULL);
   /* Connect to database */
   if (!mysql_real_connect(conn, server,
         user, password, database, 0, NULL, 0)) {
      fprintf(stderr, "%s\n", mysql_error(conn));
      exit(1);
   }
   /* send SQL query */
   if (mysql_query(conn, "show tables")) {
      fprintf(stderr, "%s\n", mysql_error(conn));
      exit(1);
   }
   res = mysql_use_result(conn);
   /* output table name */
   printf("MySQL Tables in mysql database:\n");
   while ((row = mysql_fetch_row(res)) != NULL)
      printf("%s \n", row[0]);
   /* close connection */
```

EXEMPLU: Interfete specifice MySQL
b) biblioteci dezvoltate pentru alte limbaje

MySQL Connector/ODBC
MySQL Connector/Net
MySQL Connector/J
MySQL Connector/C++
MySQL Connector/C
MySQL Connector/Python
libmysqld, the Embedded MySQL Server Library
MySQL C API
MySQL PHP API
) MySQL Perl API
I MySQL Python API
2 MySQL Ruby APIs
3 MySQL Tcl API
4 MySQL Eiffel Wrapper

EXEMPLU: Interfete independente de SGBD
c) Interfata ODBC



INTERFATA ODBC

DEFINITIE

Interfata ODBC (Open DataBase Connectivity) : **interfaţă de programare a aplicaţiilor prin apel de funcţii independente de sistemul SGBD** folosit, de SO si de limbajul de programare utilizat.

ODBC

- Componenta a Windows Open Services Architecture
- transformă apelurile de funcţii ODBC în comenzi SQL (sau într-un limbaj procedural de extensie a limbajului SQL) si le transmite la SGBD.
- permite programelor de aplicatii utilizarea instructiunilor SQL pentru a accesa date din orice baza de date indiferent de SGBD
- Exista mai multe versiuni
- Componente: ODBC Application, ODBC Driver, ODBC Driver Manager, ODBC Data Source

INTERFATA ODBC

DEFINITIE

Componente:

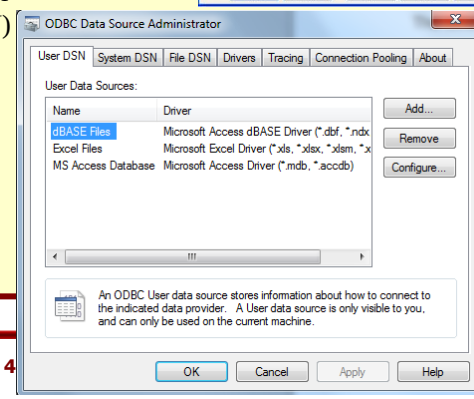
- **ODBC Application** (Front-end client) permite:
  - Definirea si realizarea aplicatiei de BD
  - Deschiderea si inchiderea sesiunii de conectare la BD
  - Apelul functiilor API care genereaza comenzile SQL
  - Definirea dimeniunii si a formatului datelor rezultate in urma interogarilor
  - Procesarea erorilor si afisarea rezultatelor
- **ODBC Driver** (Back-end server) ofera:
  - Biblioteca DLL
  - Procesarea apelului functiilor ODBC
  - Stabilirea conectarii la BD
  - Executia instructiunior SQL si returnarea rezultatului aplicatiei
  - Transforma sintaxa instructiunilor SQL din programul aplicatie in sintaxa proprie SGBD

INTERFATA ODBC

DEFINITIE

Componente: continuare

- **ODBC Driver Manager** asigura:
  - biblioteca DLL
  - legatura dintre o aplicatie ODC si un driver ODBC
  - incarca driverele necesare aplicatiei
  - asociaza "data source name" (DSN) unui driver (DLL)
- **ODBC Data Source** specifica
  - BD ce va fi accesata
  - SO asociat SGBD,SGBD, tip retea, protocol access



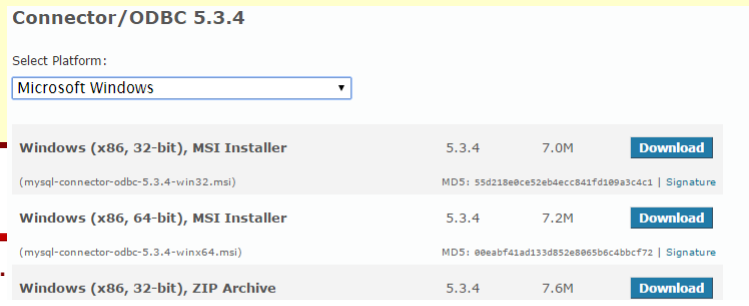
INTERFATA ODBC

DEFINITIE

Produse ODBC: Microsoft ODBC, iODBC (open source), UnixODBC, UDBC (cross-platform)

ODBC Drivers: Oracle, DB2, Microsoft SQL Server, Sybase, MySQL, Pervasive SQL, IBM Lotus Domino, FileMaker, MS Access, etc

MySQL: http://www.mysql.com/products/connector/

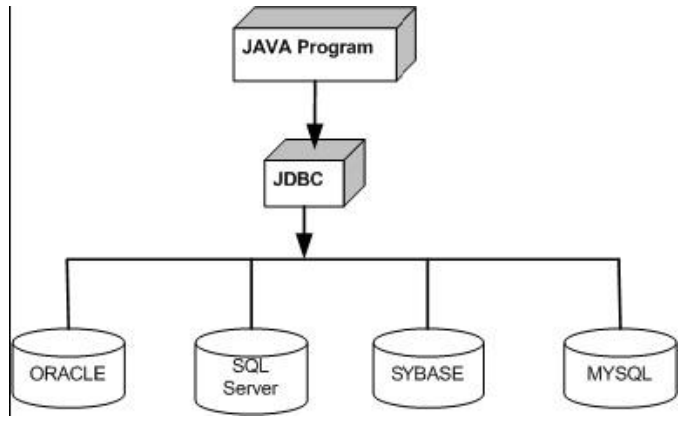


Exemplu utilizare ODBC in aplicatie PHP cu BD in MySQL

afiseaza_MYSQL.php | insert_MYSQL.php | actiune_insert_MYSQL.php

```
<html>
<head>
  <title>Afisarea datelor prin ODBC</title>
</head>
<body>
<?php
$con_id=odbc_connect("oferta_calc_dsn","root","root");
$interogare="SELECT * FROM oferta_calc";
$rez_id=odbc_do($con_id,$interogare);
echo   "<center>
<table  border=\"1\"   cellpadding=\"5\">
<tr><th>Modelul</th>
<th>Microprocesorul</th>
<th>Pretul</th></tr>";
while(odbc_fetch_row ($rez_id)) {
$model=odbc_result($rez_id,1);
$microprocesor=odbc_result($rez_id,2);
$pret = odbc_result($rez_id,3) ;
echo   "<tr><td>".$model."</td><td>".
         $microprocesor."</td> <td>".
         $pret."</td></tr>";
}
echo   "</table></center>";
echo "<a href=\"insert_MYSQL.php\">Inapoi la insert</a><br />";
odbc_close($con_id);
?>
</body>
</html>
```


EXEMPLU: Interfete independente de SGBD
d) Interfata JDBC



EXEMPLU: Interfete independente de SGBD
d) Interfata JDBC

Resures necesare pentru implementarea aplicatiilor web cu BD MySQL in Java:

Software or Resource	Version Required
NetBeans IDE	7.2, 7.3, 7.4, 8.0, Java EE bundl
Java Development Kit (JDK)	version 7 or 8
MySQL database server	5.x
MySQL Connector/J JDBC Driver	version 5.x
GlassFish Server Open Source Edition	3.x or 4.x

```
//STEP 1. Import required packages
import java.sql.*;

public class FirstExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   Statement stmt = null;
   try{
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //STEP 4: Execute a query
      System.out.println("Creating statement...");
      stmt = conn.createStatement();
      String sql;
      sql = "SELECT id, first, last, age FROM Employees";
      ResultSet rs = stmt.executeQuery(sql);
```