Indrumar – Capitolul 4

LIMBAJE SI BIBLIOTECI DE PROGRAMARE A APLICATIILOR DE BAZE DE DATE

Sistemele de gestiune a bazelor de date relationale prelucreaza instructiuni (comenzi) SQL. Limbajul SQL este un limbaj neprocedural, care permite definirea datelor si operatii de manipulare a acestora, dar nu prevede instructiuni de control al ordinii de executie a operatiilor. De aceea, pentru realizarea aplicatiilor de baze de date au fost dezvoltate o multitudine de limba je si biblioteci de programare: limbaje procedurale de extensie a limbajului SQL, limbajul SQL integrat, biblioteci de functii sau de clase pentru comunicarea cu bazele de date.

Un limbaj procedural (procedural language) este o extensie a limbajului SQL si permite combinarea instructiunilor SQL cu instructiuni de control al ordinii de executie. Astfel de limbaje sunt folosite în principal în cadrul sistemelor de gestiune, pentru stocarea unor proceduri de calcul, dar exista si posibilitatea ca programele de aplicatii sa transmita sistemului SGBD blocuri (loturi) de instructiuni într-un limbaj procedural.

Pentru dezvoltarea programelor de aplicatii de baze de date se pot aborda doua tehnologii diferite, limbajul SQL integrat într-un limbaj de nivel înalt sau interfete de programare a aplicatiilor. În limbajul SQL integrat (Embeded SQL), instructiunile limbajului SQL sunt incluse direct în codul programului sursa scris într-un limbaj gazda de nivel înalt (Ada, PL/1, Pascal, Fortran, Cobol, C). Controlul flu xului de operatii este realizat prin instructiunile limbajului gazda, iar operatiile cu baza de date sunt realizate prin instructiuni SQL.

Interfetele de programare a aplicatiilor (Application Programming Interface - API) 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.

4.1

LIMBAJE PROCEDURALE DE EXTENSIE A LIMBAJULUI SQL

Limbajele procedurale asigura controlul ordinii de executie (bucle while , instructiuni conditionale if, etc.) si, de asemenea, ofera suport de creare a cursoarelor, a procedurilor stocate , a functiilor definite de utilizator si a declansatorilor (triggere).

Un cursor (cursor) este o structura care permite memorarea (folosind un buffer în memorie) a unei multimi de linii returnate ca rezultat de o instructiune de interogare. Programele de aplicatii nu pot prelucra deodata toate liniile rezultatului si folosesc cursoare pentru extragerea individuala a unei linii (sau a unui grup de linii) din multimea de linii rezultat. În fiecare moment, într-un cursor exista o pozitie curenta (linie curenta) în multimea de linii rezultat. La fiecare operatie de extragere, se citesc una sau mai multe linii relativ la pozitia curenta a cursorului, iar aceasta pozitie se actualizeaza conform modului de parcurgere (înainte sau înapoi). Cursoarele se pot crea atât la nivelul limbajului SQL2 sau a extensiilor procedurale ale acestuia, cât si prin intermediul limbajului SQL integrat (Embedded SQL) sau a bibliotecilor si interfetelor de programare a aplicatiilor de baze de date (ca, de exemplu, interfetele ODBC si JDBC).

O procedura stocata (stored procedure) este o procedura care implementeaza o parte din

algoritmii de calcul ai aplicatiilor si este memorata în baza de date la fel ca si alte obiecte ale bazei de date. Procedurile stocate sunt compilate si optimizate de sistemul de gestiune o singura data, atunci când sunt folosite prima oara si ramân memorate în server pentru oricâte apeluri ulterioare.

O functie definita de utilizator (user-defined function) este o functie memorata în baza de date, la fel ca o procedura stocata; diferenta între acestea este ca o functie returneaza întotdeauna o valoare si poate fi folosita direct în expresii, pe câta vreme o procedura stocata poate sa nu returneze nici o valoare.

Un trigger este o procedura stocata cu o functionare speciala, care se declanseaza automat atunci când se efectueaza o operatie de actualizare a unei relatii. Prin triggere se pot specifica si impune procedural constrângerile explicite, cum sunt dependentele de date care nu sunt determinate de chei ale relatiilor. De asemenea, triggerele mai sunt folosite si pentru generarea automata a unor valori care rezulta din valori ale altor atribute, precum si pentru jurnalizarea transparenta a evenimentelor sau culegerea de date statistice în legatura cu accesarea relatiilor.

Majoritatea sistemelor de gestiune sunt prevazute cu cel putin un limbaj procedural, cele mai cunoscute fiind: PL/SQL pentru sistemele de gestiune Oracle, Transact-SQL pentru sistemele de gestiune Microsoft SQL Server, MySql, PL/PGSQL si PL/Pearl pentru sistemul de gestiune PostgreSQL, etc.

4.1.1 LIMBAJUL TRANSACT-SQL

Programele Transact-SQL pot fi executate ca proceduri memorate în cadrul bazei de date sau ca loturi (batchs) de instructiuni transmise sistemului prin intermediul unei aplicatii sau a unui program utilitar de acces interactiv la baza de date. Programele utilitare de acces interactiv din distributia SQL Server sunt isql sau osql (la nivel de linie de comanda) si SQL Query Analizer (care ofera si o interfata grafica). De asemenea, numeroase operatii de definire a tabelelor si a altor caracteristici ale bazelor de date se pot face din consola de administrare (Enterprise Manager) a sistemului SQL Server. La fel se lucreaza si cu SGBD-ul MySql unde se poate lucra din terminal in linux sau din utilitare precum MySql Workbench.

4.1.1.1 Elementele de baza ale limbajului Transact-SQL

Loturi de prelucrare. Un lot (batch) consta dintr-o secventa de instructiuni Transact-SQL terminata cu comanda GO (in SQL Server). Exista mai multe reguli de grupare a instructiunilor în loturi. De exemplu: orice instructiune CREATE trebuie sa fie prima în lot; de aceea nu pot fi grupate în acelasi lot instructiunile CREATE TABLE, CREATE VIEW, CREATE TRIGGER , CREATE RULE, etc.

Variabile locale. În limbajul Transact-SQL pot fi folosite variabile locale pentru memorarea unor valori care pot fi testate sau modificate (ca în orice alt limbaj) si, în plus, asigura transferul datelor catre si de la tabelele bazei de date. Variabilele locale au ca domeniu de definitie lotul, procedura sau blocul în care au fost declarate.

O variabila locala se declara folosind instructiunea DECLARE care specifica un identificator si tipul variabilei.

Sintaxa de declarare arata astfel:

DECLARE nume_variabila tip_date

Limbajul Transact-SQL suporta toate tipurile de date prevazute în standardul SQL2 si, în plus, permite definirea de catre utilizator a unor noi tipuri de date. Initializarea variabilelor locale se poate face printr-o comanda SELECT (sau SET), cu urmatoarea sintaxa:

SELECT @nume_variabila = expresie

Instructiuni SQL. Limbajul Transact-SQL suporta toate instructiunile SQL, cu unele modificari

de sintaxa, astfel încât sa poata fi folosite în combinatie cu variabilele locale ale programului. De exemplu, forma modificata a instructiunii SELECT , prin care se asigneaza variabile locale cu valori ale unor atribute selectate, este:
SELECT @var1 = col1, @var2 = col2, ... @varn = coln FROM lista_tabele WHERE conditie (in SQL Server)
In MySql sintaxa este diferita:
SELECT col1 , col2 into var1, var2 FROM CURSE WHERE conditie
Atentie: tipurile coloanelor selectate trebuie sa respecte aceeasi ordine cu tipurile variabilelor in care se pun aceste valori.

O astfel de instructiune este utila pentru interogarile care returneaza o singura linie, deoarece variabilele locale sunt setate cu valorile coloanelor primei linii a rezultatului, iar valorile din celelalte linii se pierd (nemaifiind loc unde sa fie memorate). Atunci când o interogare returneaza o multime de linii se poate folosi un cursor, asa cum va fi prezentat în continuare. Instructiuni de control a ordinii de executie. Ordinea de executie a instructiunilor unui lot sau a unei proceduri stocate este controlata prin urmatoarele instructiunile de control:
BEGIN...END
GOTO
IF...ELSE
RETURN
WAITFOR
WHILE
BREAK
CONTINUE

Semnificatia acestor instructiuni este cea generala, cunoscuta din alte limbaje de programare, cu anumite particularitati. Instructiunile de control nu pot depasi granitele unei lot de executie sau a unei proceduri stocate.

4.1.1.2 Cursoare Transact-SQL

În limbajul Transact-SQL se pot defini cursoare cu urmatoarea instructiune:
DECLARE nume_cursor CURSOR [optiuni] FOR instructiune_select

Exista mai multe optiuni care se pot seta si care definesc tipul cursorului. Semnificatia acestor optiuni este descrisa în manualul sistemului (Books Online), iar în programul care urmeaza este dat un exemplu de creare si utilizare a unui cursor Transact-SQL. La deschiderea unui cursor (prin comanda OPEN) se executa interogarea respectiva (instructiunea SELECT) si rezultatul (multimea de linii) se încarca în cursor. Dupa încarcarea cursorului, se pot extrage liniile prin operatii de extragere cu instructiunea FETCH, a carei sintaxa simplificata arata astfel:

FETCH [[NEXT|PRIOR|FIRST|LAST|RELATIVE n|ABSOLUTE n] FROM] @nume_cursor [INTO @var1,@var2,...@varn]]

Clauza INTO permite extragerea valorilor unei linii în variabilele locale. Asocierea dintre variabilele locale si coloanele cursorului este implicit pozitionala: valoarea din prima coloana se înscrie în prima variabila (@var1), valoarea din a doua coloana se înscrie în variabila a doua (@var2), etc. Ordinea coloanelor cursorului este ordinea coloanelor din instructiunea SELECT asociata cursorului. Pentru testarea starii unui cursor se apeleaza functia globala @@FETCH_STATUS(in SQL Server), care raporteaza starea ultimei instructiuni FETCH executate pentru conexiunea curenta la server. Valoarea returnata este 0, daca extragerea a fost efectuata cu succes, 1, daca a survenit o eroare si 2, daca nu mai sunt date în multimea de linii rezultat. Dupa terminarea operatiilor asupra datelor

cursorului, acesta se închide cu comanda CLOSE, iar instructiunea DEALLOCATE sterge structurile de date ale cursorului si elibereaza memoria.

Un exemplu de trigger ce contine un cursor poate fi urmatorul:

```
-- Trigger DDL Statements
DELIMITER $$

USE `lab5`$$

CREATE
DEFINER=`root`@`localhost`
TRIGGER `lab5`.`tg_AFS`
AFTER INSERT ON `lab5`.`AFS`
FOR EACH ROW
BEGIN
– Blocul declaratiilor
DECLARE v_functia varchar(20);
DECLARE v_salariu decimal;
DECLARE n_functia varchar(20);
DECLARE n_salariu decimal;
DECLARE mesaj varchar(20);
DECLARE no_more_rows BOOLEAN;

-- Crearea unui cursor
DECLARE cursor_AFS CURSOR
FOR
SELECT Functie, Salariu FROM AFS;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_rows = TRUE;

SET n_functia = new.Functie, n_salariu = new.Salariu;

OPEN Cursor_AFS;

-- Parcurgerea liniilor tabelului AP folosind cursorul
the_loop: LOOP
        FETCH cursor_AFS INTO v_functia, v_salariu;
        IF n_functia = v_functia AND n_salariu != v_salariu THEN
                        set mesaj = "Eroare DF";
                        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = mesaj;
                ELSE
                        IF no_more_rows THEN
                                LEAVE the_loop;
                        END IF;
                END IF;
END LOOP the_loop;

CLOSE cursor_AFS;
END$$
```

4.2 . INTERFETE DE PROGRAMARE A APLICATIILOR DE BAZE DE DATE

Interfetele de programare a aplicatiilor (API) reprezinta cea mai cunoscuta tehnica de dezvoltare a aplicatiilor de baze de date, fiind mult mai utilizata decât tehnica de programare în limbajul SQL integrat, deoarece programele rezultate sunt mai flexibile si mai usor de dezvoltat si de întretinut. Ca interfete de programare a bazelor de date, exista atât interfete specifice, oferite de diferite sisteme SGBD, cât si interfete cu un grad mare de generalitate, care pot fi folosite pentru mai multe tipuri de sisteme SGBD, cum sunt interfetele ODBC (Open Database Connectivity) sau JDBC (Java Database Connectivity).

In prezenta lucrare se va implementa o aplicatie simpla ce foloseste tehnologiile Java Server Pages si JavaBean pentru a accesa o tabela implementata in MySql.

Pachetele software necesare dezvoltarii unei astfel de aplicatii sunt:

1. Serverul de MySql;
2. MySql Workbench (nu este obligatoriu – toate operatiile de creare a bazelor de date pot fi efectuate si in terminal);
3. Platforma Eclipse JEE (se descarca de pe siteul www.eclipse.org ultima versiune); arhiva se dezarhiveaza intr-un director de lucru.
4. Serverul Apache Tomcat (se descarca de pe site ultima versiune); arhiva va fi dezarhivata in vederea legarii serverului Apache Tomcat la platforma Eclipse.
5. Conectorul mysql pentru Java (se descarca ultima versiune).

Aplicatia trebuie sa permita operatii simple de vizualizare/stergere/modificare a datelor din tabela.

Etapele dezvoltarii unei interfete la o baza de date MySql

Pasul 1 - Crearea bazei de date in MySql

Baza de date se va numi “Clinica” (atentie: linux-ul este case sensitive – respectati intocmai denumirile date).

Se creaza tabela “PACIENTI” cu urmatoarele campuri:

- IdPacient, de tip BIGINT (PrimaryKey, Auto_increment);
- CNP de tip VARCHAR(45);
- NumePacient de tip VARCHAR(45);
- PrenumePacient de tip VARCHAR(45);
- Varsta de tip VARCHAR(45);
- Adresa de tip VARCHAR(45);
- Email de tip VARCHAR(45);
- Categorie de tip VARCHAR(45);
- Telefon de tip VARCHAR(45).

Pasul 2 – configurare Eclipse

Se intra in meniul Window->Preferences->Data Management->Connectivity->Driver Definition. Se apasa butonul Add. Se alege MySql – ultima versiune disponibila. Se da Remove la jar-ul existent si se adauga jar-ul din cadrul conectorului mysql descarcat anterior.

Se creaza o aplicatie de tip Server si se face legatura cu serverul Apache Tomcat versiunea 7 (File->New->Other->Server). Din lista serverelor posibile se alege Tomcat versiunea 7: se da calea

catre serverul apache descarcat anterior.

Se creaza o aplicatie de tip Dynamic Web Application.

Legarea proiectului de conectorul mysql se face în felul următor:

- click dreapta pe proiectul nou creat (în Project Explorer);
- Build Path->Configure Build Path;
- În cadrul opțiunii Java Build Path se selectează din bara de meniuri “Libraries”;
- Din partea dreaptă a ferestrei se apasă butonul Add Library;
- Se alege prima opțiune – Connectivity Driver Definition;
- Din drop-down-ul ferestrei noi se alege driverul JDBC pentru MySql (MySql JDBC Driver) după care se apasă Finish;
- Din partea stângă a ferestrei se alege opțiunea Deployment Assembly. Va apare o fereastră pentru aplicarea setărilor efectuate până în acest moment – se alege Apply;
- La opțiunea Deployment Assembly se apasă butonul Add din partea dreaptă;
- Se alege Java Build Path Entries->MySql JDBC Driver->Finish;
- Se apasă Apply și Ok din fereastra principală.

În proiectul creat vor fi doua module importante: paginile web implementate cu tehnologia Java Server Pages si o clasa javabean care va contine toate functiile ce vor opera cu datele din baza de date: functii de conectare, deconectare, interogare a bazei de date, stergere si modificare.

Pe proiect se da click dreapta si se alege crearea unei clase java clasice cu numele “javabean” - aceasta va fi pusa intr-un pachet numit “db”.

Sectiunea urmatoare reprezinta intregul continut al clasei “javabean”:

```
package db;

import java.sql.*;

public class javabean {
	String error;
	Connection con;

	public javabean(){}

	public void connect() throws ClassNotFoundException, SQLException, Exception
	{
		try
		{
			Class.forName("com.mysql.jdbc.Driver");
		   con = DriverManager.getConnection("jdbc:mysql://localhost:3306/Clinica","root", "us_125a;");
		}
		catch (ClassNotFoundException cnfe)
		{
			error = "ClassNotFoundException: Nu s-a gasit driverul bazei de date.";
			throw new ClassNotFoundException(error);
		}
		catch (SQLException cnfe)
		{
			error = "SQLException: Nu se poate conecta la baza de date.";
			throw new SQLException(error);
		}
		catch (Exception e)
		{
			error = "Exception: A aparut o exceptie neprevazuta in timp ce se stabilea legatura la baza de date.";
			throw new Exception(error);
		}
	} // connect()

	public void connect(String bd) throws ClassNotFoundException, SQLException, Exception
	{
```

```
		try
		{
			Class.forName("com.mysql.jdbc.Driver");
		   con = DriverManager.getConnection("jdbc:mysql://localhost:3306/"+bd,"root", "us_125a;");
		}
		catch (ClassNotFoundException cnfe)
		{
			error = "ClassNotFoundException: Nu s-a gasit driverul bazei de date.";
			throw new ClassNotFoundException(error);
		}
		catch (SQLException cnfe)
		{
			error = "SQLException: Nu se poate conecta la baza de date.";
			throw new SQLException(error);
		}
		catch (Exception e)
		{
			error = "Exception: A aparut o exceptie neprevazuta in timp ce se stabilea legatura la baza de date.";
			throw new Exception(error);
		}
	} // connect(String bd)

	public void connect(String bd, String ip) throws ClassNotFoundException, SQLException, Exception
	{
		try
		{
			Class.forName("com.mysql.jdbc.Driver");
		   con = DriverManager.getConnection("jdbc:mysql://"+ip+":3306/"+bd,"root", "us_125a;");
		}
		catch (ClassNotFoundException cnfe)
		{
			error = "ClassNotFoundException: Nu s-a gasit driverul bazei de date.";
			throw new ClassNotFoundException(error);
		}
		catch (SQLException cnfe)
		{
			error = "SQLException: Nu se poate conecta la baza de date.";
			throw new SQLException(error);
		}
		catch (Exception e)
		{
			error = "Exception: A aparut o exceptie neprevazuta in timp ce se stabilea legatura la baza de date.";
			throw new Exception(error);
		}
	} // connect(String bd, String ip)

	public void disconnect() throws SQLException
	{
		try
		{
			if ( con != null )
			{
				con.close();
			}
		}
		catch (SQLException sqle)
		{
			error = ("SQLException: Nu se poate inchide conexiunea la baza de date.");
			throw new SQLException(error);
		}
	} // disconnect()

	public void adaugaPacient(String CNP, String NumePacient, String PrenumePacient, String Varsta, String Adresa, String Email,
String Categorie, String Telefon)
			throws SQLException, Exception
```

```
                                {
                                 if (con != null)
                                 {
                                  try
                                  {
                                   // create a prepared SQL statement
                                   Statement stmt;
                                   stmt = con.createStatement();
                                   stmt.executeUpdate("insert into `Clinica`.`PACIENTI`(CNP, NumePacient, PrenumePacient, Varsta,
Adresa, Email, Categorie, Telefon) values('"+CNP+"', '"+NumePacient+"', '"+PrenumePacient+"', '"+Varsta+"', '"+Adresa+"', '"+Email+"',
'"+Categorie+"', '"+Telefon+"');");
                                  }
                                  catch (SQLException sqle)
                                  {
                                   error = "ExceptieSQL: Reactualizare nereusita; este posibil sa existe duplicate.";
                                   throw new SQLException(error);
                                  }
                                 }
                                 else
                                 {
                                  error = "Exceptie: Conexiunea cu baza de date a fost pierduta.";
                                  throw new Exception(error);
                                 }
                                } // end of adaugaPacient()

        public ResultSet vedeTabela(String tabel) throws SQLException, Exception
         {
                ResultSet rs = null;
                try
                {
                        String queryString = ("select * from `Clinica`.`" + tabel + "`;");
                        Statement stmt = con.createStatement(/*ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY*/);
                        rs = stmt.executeQuery(queryString);
                        //System.out.println("111**********************************************************************
*****");
                }
                catch (SQLException sqle)
                {
                        error = "SQLException: Interogarea nu a fost posibila.";
                        throw new SQLException(error);
                }
                catch (Exception e)
                {
                        error = "A aparut o exceptie in timp ce se extrageau datele.";
                        throw new Exception(error);
                }
                return rs;
         } // vedeTabela()

        public void stergeDateTabela(String[] primaryKeys, String tabela, String dupaID) throws SQLException, Exception
         {
          if (con != null)
          {
           try
           {
            // create a prepared SQL statement
            long aux;
            PreparedStatement delete;
            delete = con.prepareStatement("DELETE FROM "+ tabela + " WHERE " + dupaID +"=?;");
            for (int i = 0; i < primaryKeys.length; i++)
            {
             aux=java.lang.Long.parseLong(primaryKeys[i]);
             delete.setLong(1, aux);
             delete.execute();
```

```
            }
           }
           catch (SQLException sqle)
           {
            error = "ExceptieSQL: Reactualizare nereusita; este posibil sa existe duplicate.";
            throw new SQLException(error);
           }
           catch (Exception e)
           {
            error = "A aparut o exceptie in timp ce erau sterse inregistrarile.";
            throw new Exception(error);
           }
          }
          else
          {
           error = "Exceptie: Conexiunea cu baza de date a fost pierduta.";
           throw new Exception(error);
          }
         } // end of stergeDateTabela()
        public void stergeTabela(String tabela) throws SQLException, Exception
        {
                if (con != null)
                {
                 try
                 {
                  // create a prepared SQL statement
                  Statement stmt;
                  stmt = con.createStatement();
                  stmt.executeUpdate("delete from " + tabela + ";");
                 }
                 catch (SQLException sqle)
                 {
                  error = "ExceptieSQL: Stergere nereusita; este posibil sa existe duplicate.";
                  throw new SQLException(error);
                 }
                }
                else
                {
                 error = "Exceptie: Conexiunea cu baza de date a fost pierduta.";
                 throw new Exception(error);
                }
        } // end of stergeTabela()

        public void modificaTabela(String tabela, String IDTabela, long ID, String[] campuri, String[] valori)
throws SQLException, Exception
        {
                        String update="update " + tabela + " set ";
                        String temp="";
         if (con != null)
         {
          try
          {
                for(int i=0; i<campuri.length; i++){
                        if(i!=(campuri.length-1))
                                temp = temp + campuri[i] + "='"+ valori[i] +"', ";
                        else
                                temp = temp + campuri[i] + "='"+ valori[i] +"' where " + IDTabela + " = '" + ID + "';";
                }
                update = update + temp;
           // create a prepared SQL statement
           Statement stmt;
           stmt = con.createStatement();
           stmt.executeUpdate(update);
          }
          catch (SQLException sqle)
```

```
            {
              error = "ExceptieSQL: Reactualizare nereusita; este posibil sa existe duplicate.";
              throw new SQLException(error);
            }
          }
          else
          {
            error = "Exceptie: Conexiunea cu baza de date a fost pierduta.";
            throw new Exception(error);
          }
        } // end of modificaTabela()

        public ResultSet intoarceLinie(String tabela, long ID) throws SQLException, Exception
        {
          ResultSet rs = null;
          try
          {
            // Execute query
            String queryString = ("SELECT * FROM " + tabela + " where BoalaID=" + ID + ";");
            Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
            rs = stmt.executeQuery(queryString); //sql exception
          }
          catch (SQLException sqle)
          {
            error = "SQLException: Interogarea nu a fost posibila.";
            throw new SQLException(error);
          }

          catch (Exception e)
          {
            error = "A aparut o exceptie in timp ce se extrageau datele.";
            throw new Exception(error);
          }
          return rs;
        } // end of intoarceLinie()

        public ResultSet intoarceLinieDupaId(String tabela, String denumireId, long ID) throws SQLException, Exception
        {
          ResultSet rs = null;
          try
          {
            // Execute query
            String queryString = ("SELECT * FROM " + tabela + " where "+ denumireId +"=" + ID + ";");
            Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
            rs = stmt.executeQuery(queryString); //sql exception
          }
          catch (SQLException sqle)
          {
            error = "SQLException: Interogarea nu a fost posibila.";
            throw new SQLException(error);
          }

          catch (Exception e)
          {
            error = "A aparut o exceptie in timp ce se extrageau datele.";
            throw new Exception(error);
          }
          return rs;
        } // end of intoarceLinieDupaId()


}
```

Se creaza pagina principala index.jsp (File->New->Other->Web->JSP File):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

```
<%@ page language="java" import="java.lang.*,java.math.*,db.*,java.sql.*, java.io.*, java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>index</title>
</head>
<body>
<hr>
<hr>
<br/>
<h2>Vizualizari + Adaugari + Stergeri</h2>
<p> <a href="tabela_Pacienti.jsp"><b><strong>Pacienti</strong></b></a></p>
<br/>
<hr>
<ul>
		<li><a href="modifica_Pacient.jsp"><b><strong>Modifica Pacienti</strong></b></a></li>
</ul>
<hr>
<hr>
<br/>
</body>
</html>
```

Pagina de vizualizare a tabelei PACIENTI – tabela_Pacienti.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page language="java" import="java.lang.*,java.math.*,db.*,java.sql.*, java.io.*, java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tabela Pacienti</title>
</head>

<jsp:useBean id="jb" scope="session" class="db.javabean" />
<jsp:setProperty name="jb" property="*" />

<body>
<h1 align="center"> Tabela Pacienti:</h1>
<br/>
<p align="center"><a href="nou_Pacient.jsp"><b>Adauga un nou pacient.</b></a>  <a href="index.jsp"><b>Home</b></a></p>
<form action="sterge_Pacient.jsp" method="post">
<table border="1" align="center">
<tr>
<td><b>Mark:</b></td>
<td><b>IdPacient:</b></td>
<td><b>CNP:</b></td>
<td><b>Nume Pacient:</b></td>
<td><b>Prenume Pacient:</b></td>
<td><b>Varsta:</b></td>
<td><b>Adresa:</b></td>
<td><b>Email:</b></td>
<td><b>Categorie:</b></td>
<td><b>Telefon:</b></td>
</tr>
<%		jb.connect();
		ResultSet rs = jb.vedeTabela("PACIENTI");
		long x;
		while (rs.next())  {
			x=rs.getLong("IdPacient");
%>
<tr>
<td><input type="checkbox" name="primarykey" value="<%= x %>" /></td>
```

```
<td><%= x %></td>
<td><%= rs.getString("CNP") %></td>
<td><%= rs.getString("NumePacient") %></td>
<td><%= rs.getString("PrenumePacient") %></td>
<td><%= rs.getString("Varsta") %></td>
<td><%= rs.getString("Adresa") %></td>
<td><%= rs.getString("Email") %></td>
<td><%= rs.getString("Categorie") %></td>
<td><%= rs.getString("Telefon") %></td>
<%		}
%>
</tr>
</table><br/>
<p align="center">
<input type="submit" value="Sterge liniile marcate">
</p>
</form>

<%
rs.close();
jb.disconnect();
%>

<br/>

<p align="center">
<a href="index.jsp"><b>Home</b></a>
<br/>
</p>
</body>
</html>
```

Pagina de adaugare a unui nou pacient – nou_Pacient.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page language="java" import="java.lang.*,java.math.*,db.*,java.sql.*, java.io.*, java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Adauga pacient</title>
</head>

<jsp:useBean id="jb" scope="session" class="db.javabean" />
<jsp:setProperty name="jb" property="*" />

<body>

<%
String CNP =  request.getParameter("CNP");
String NumePacient =  request.getParameter("NumePacient");
String PrenumePacient =  request.getParameter("PrenumePacient");
String Varsta =  request.getParameter("Varsta");
String Adresa =  request.getParameter("Adresa");
String Email=  request.getParameter("Email");
String Categorie =  request.getParameter("Categorie");
String Telefon =  request.getParameter("Telefon");

	if(CNP!=null){
		jb.connect();
		jb.adaugaPacient(CNP, NumePacient, PrenumePacient, Varsta, Adresa, Email, Categorie, Telefon);
		jb.disconnect();
		%>
<p>Datele au fost adaugate.</p>
```

```
                    <%
            }
            else{

%>

<h1> Suntem in tabela Pacient.</h1>

<form action="nou_Pacient.jsp" method="post">
<table>

<tr>
            <td align="right">CNP:</td>
            <td> <input type="text" name="CNP" size="40" /></td>
</tr>

<tr>
            <td align="right">Nume Pacient:</td>
            <td> <input type="text" name="NumePacient" size="40" /></td>
</tr>

<tr>
            <td align="right">Prenume Pacient:</td>
            <td> <input type="text" name="PrenumePacient" size="30" /></td>
</tr>

<tr>
            <td align="right">Varsta:</td>
            <td> <input type="text" name="Varsta" size="30" /></td>
</tr>

<tr>
            <td align="right">Adresa:</td>
            <td> <input type="text" name="Adresa" size="30" /></td>
</tr>

<tr>
            <td align="right">Email:</td>
            <td> <input type="text" name="Email" size="30" /></td>
</tr>

<tr>
            <td align="right">Categorie:</td>
            <td> <input type="text" name="Categorie" size="30" /></td>
</tr>

<tr>
            <td align="right">Telefon:</td>
            <td> <input type="text" name="Telefon" size="30" /></td>
</tr>

</table>

<input type="submit" value="Adauga pacientul" />

</form>
<%
            }
%>
<br/>
<a href="index.jsp"><b>Home</b></a>
<br/>
</body>
</html>
```

Pagina de stergere a pacientilor – sterge_Pacient.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page language="java" import="java.lang.*,java.math.*,db.*,java.sql.*, java.io.*, java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tabela Pacienti</title>
</head>

<jsp:useBean id="jb" scope="session" class="db.javabean" />
<jsp:setProperty name="jb" property="*" />

<body>
<%
	String[] s = request.getParameterValues("primarykey");
	jb.connect();
	jb.stergeDateTabela(s, "PACIENTI", "IdPacient");
	jb.disconnect();
%>

<p align="center">
<a href="index.jsp"><b>Home</b></a>
<br/>
</p>
</body>
</html>
```

Pagina de modificare a datelor unui pacient – modifica_Pacient.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page language="java" import="java.lang.*,java.math.*,db.*,java.sql.*, java.io.*, java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tabela Pacienti</title>
</head>

<jsp:useBean id="jb" scope="session" class="db.javabean" />
<jsp:setProperty name="jb" property="*" />

<body>

<h1 align="center"> Tabela Pacienti:</h1>
<br/>
<p align="center"><a href="nou_Pacient.jsp"><b>Adauga un nou pacient.</b></a>  <a href="clinica.jsp"><b>Home</b></a></p>
<form action="m1_Pacient.jsp" method="post">
<table border="1" align="center">
<tr>
<td><b>Mark:</b></td>
<td><b>IdPacient:</b></td>
<td><b>CNP:</b></td>
<td><b>Nume Pacient:</b></td>
<td><b>Prenume Pacient:</b></td>
<td><b>Varsta:</b></td>
<td><b>Adresa:</b></td>
<td><b>Email:</b></td>
<td><b>Categorie:</b></td>
<td><b>Telefon:</b></td>
</tr>
<%	jb.connect();
	ResultSet rs = jb.vedeTabela("PACIENTI");
```

```
		long x;
		while (rs.next())  {
			x=rs.getLong("IdPacient");
%>
<tr>
<td><input type="checkbox" name="primarykey" value="<%= x %>" /></td>
<td><%= x %></td>
<td><%= rs.getString("CNP") %></td>
<td><%= rs.getString("NumePacient") %></td>
<td><%= rs.getString("PrenumePacient") %></td>
<td><%= rs.getString("Varsta") %></td>
<td><%= rs.getString("Adresa") %></td>
<td><%= rs.getString("Email") %></td>
<td><%= rs.getString("Categorie") %></td>
<td><%= rs.getString("Telefon") %></td>
<%		}
%>
</tr>
</table><br/>
<p align="center">
<input type="submit" value="Modifica linia">
</p>
</form>

<%		jb.disconnect(); %>

<br/>

<p align="center">
<a href="index.jsp"><b>Home</b></a>
<br/>
</p>
</body>
</html>
```

Pagina de preluare a datelor initiale ale pacientului care va suferi modificari – m1_Pacient.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page language="java" import="java.lang.*,java.math.*,db.*,java.sql.*, java.io.*, java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tabela Pacienti</title>
<link href="table.css" rel="stylesheet" type="text/css" media="screen" />
</head>

<jsp:useBean id="jb" scope="session" class="db.javabean" />
<jsp:setProperty name="jb" property="*" />

<body>

<h1 align="center">Tabela Pacienti:</h1>
<br/>
<p align="center"><a href="nou_Pacient.jsp"><b>Adauga un nou pacient.</b></a>  <a href="clinica.jsp"><b>Home</b></a></p>

<%

jb.connect();
long aux = java.lang.Long.parseLong(request.getParameter("primarykey"));
ResultSet rs = jb.intoarceLinieDupaId("PACIENTI", "IdPacient", aux);
rs.first();
String CNP = rs.getString("CNP");
String NumePacient = rs.getString("NumePacient");
```

```
String PrenumePacient = rs.getString("PrenumePacient");
String Varsta = rs.getString("Varsta");
String Adresa = rs.getString("Adresa");
String Email = rs.getString("Categorie");
String Categorie = rs.getString("Categorie");
String Telefon = rs.getString("Telefon");
rs.close();
jb.disconnect();

%>

<form action="m2_Pacient.jsp" method="post">
<table align="center"">

<tr>
	<td align="right">IdPacient:</td>
	<td> <input type="text" name="IdPacient" size="30" value="<%= aux %>" readonly/></td>
</tr>

<tr>
	<td align="right">CNP:</td>
	<td> <input type="text" name="CNP" size="30" value="<%= CNP %>"/></td>
</tr>

<tr>
	<td align="right">NumePacient:</td>
	<td> <input type="text" name="NumePacient" size="30" value="<%= NumePacient %>"/></td>
</tr>

<tr>
	<td align="right">PrenumePacient:</td>
	<td> <input type="text" name="PrenumePacient" size="30" value="<%= PrenumePacient %>"/></td>
</tr>

<tr>
	<td align="right">Varsta:</td>
	<td> <input type="text" name="Varsta" size="30" value="<%= Varsta %>"/></td>
</tr>

<tr>
	<td align="right">Adresa:</td>
	<td> <input type="text" name="Adresa" size="30" value="<%= Adresa %>"/></td>
</tr>

<tr>
	<td align="right">Varsta:</td>
	<td> <input type="text" name="Varsta" size="30" value="<%= Varsta %>"/></td>
</tr>

<tr>
	<td align="right">Email:</td>
	<td> <input type="text" name="Email" size="30" value="<%= Email %>"/></td>
</tr>

<tr>
	<td align="right">Categorie:</td>
	<td> <input type="text" name="Categorie" size="30" value="<%= Categorie %>"/></td>
</tr>

<tr>
	<td align="right">Telefon:</td>
	<td> <input type="text" name="Telefon" size="30" value="<%= Telefon %>"/></td>
</tr>

</table>
```

```
<p align="center">
<input type="submit" value="Modifica linia">
</p>
</form>
<p align="center"">
<a href="index.jsp"><b>Home</b></a>
<br/>
</body>
</html>
```

Pagina in care se va apela functia de modificare din javabean – m2_Pacient.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page language="java" import="java.lang.*,java.math.*,db.*,java.sql.*, java.io.*, java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tabela Pacient</title>
<link href="table.css" rel="stylesheet" type="text/css" media="screen" />
</head>

<jsp:useBean id="jb" scope="session" class="db.javabean" />
<jsp:setProperty name="jb" property="*" />

<body>

<h1 align="center"> Tabela Pacienti:</h1>
<br/>
<p align="center"><a href="nou_Pacient.jsp"><b>Adauga un nou pacient.</b></a>  <a href="index.jsp"><b>Home</b></a></p>

<%

jb.connect();
long aux = java.lang.Long.parseLong(request.getParameter("IdPacient"));
String CNP = request.getParameter("CNP");
String NumePacient = request.getParameter("NumePacient");
String PrenumePacient = request.getParameter("PrenumePacient");
String Varsta = request.getParameter("Varsta");
String Adresa = request.getParameter("Adresa");
String Email = request.getParameter("Email");
String Categorie = request.getParameter("Categorie");
String Telefon = request.getParameter("Telefon");
String[] valori = {CNP, NumePacient, PrenumePacient, Varsta, Adresa, Email, Categorie, Telefon};
String[] campuri = {"CNP", "NumePacient", "PrenumePacient", "Varsta", "Adresa", "Email", "Categorie", "Telefon"};
jb.modificaTabela("PACIENTI", "IdPacient", aux, campuri, valori);
jb.disconnect();

%>
<h1 align="center">Modificarile au fost efectuate !</h1>
<p align="center"">
<a href="index.jsp"><b>Home</b></a>
<br/>
</body>
</html>
```