A picture containing diagram

Description automatically generated**Universitatea Politehnica din București**

Facultatea de Electronică, Telecomunicații și

Tehnologia Informației

Proiect

**Programarea interfețelor pentru baze de date**

Profesor coordonator:

Dr. ing. Valentin Pupezescu

Arușilor Diana Nicoleta

Grupa 433C

București, 2021

1. **Cerință temă**

Creaţi două aplicaţii care să conţină o bază de date creată în sistemul de gestiune a bazelor de date MySql şi două interfeţe la aceasta (baza de date este comună). La crearea interfeţelor se vor folosi două tehnologii(la alegere - ex.: PHP, JSP, Hibernate, JPA, .NET etc.). Baza de date va fi compusă din tabelele stabilite în lista de teme:

Mașini M:1 Producători 1:N LocațieFabrică

1. **Descrierea tehnologiilor folosite**

Baza de date a fost realizată cu ajutorul programului MySQL Workbench, care este un instrument vizual de proiectare a bazelor de date ce integrează dezvoltarea, administrarea, proiectarea bazelor de date, crearea și întreținerea SQL într-un singur mediu de dezvoltare integrat pentru sistemul de baze de date MySQL. [1]

Cele două tehnologii folosite în crearea aplicațiilor sunt Java Server Pages (JSP) și JavaFX.

**Java Server Pages (JSP, Jakarta Server Pages)** este o tehnologie folosită la crearea de interfețe Web pentru aplicațiile care rulează pe platforma Java, creată de Sun. Ea se bazează pe tehnologia numită Java Servlets fiind, de fapt, o completare a acesteia în ideea creării cât mai facile a paginilor Web dinamice,  furnizând o modalitate de a separa partea de procesare de cea de prezentare.[1]

Punctul central al tehnologiei este reprezentat de paginileJSP care sunt fișiere text ce combină descrieri HTML cu cod Java. Acestea sunt create să suporte mai multe tipuri de documente structurate, îndeosebi HTML și XML. În general, JSP-urile folosesc anumite informații pe care le trimit la server printr-o cerere HTTP care interacționează cu datele existente pe acesta și creează dinamic un răspuns organizat într-un format standard (HTML, DHTL, XML, etc) sau într-un format text neorganizat ce va fi trimis înapoi clientului.[2] Toate prelucrările legate de paginile JSP se fac pe partea de server, acestea nefiind transmise în forma originală către client. Astfel, paginile JSP sunt gestionate și accesibile prin intermediul unui server de aplicatii, acesta incluzând și o masină virtuală Java în care rulează atât codul Java întâlnit în paginile JSP, cât și obiectele instanțiate de acesta.

Un compilator JavaServer Pages este un program care analizează JSP-urile și le transformă în servleturi Java executabile. Un program de acest tip este de obicei incorporat în serverul de aplicații și rulează automat prima dată când se accesează un JSP, dar paginile pot fi, de asemenea, precompilate pentru o performanță mai bună sau compilate ca parte a procesului de construire pentru a testa erorile. Pentru a implementa și rula Jakarta Server Pages, este necesar un server web compatibil cu un container servlet, cum ar fi Apache Tomcat sau Jetty.

În general, aplicatie Web are la bază Java Runtime Environment (ce va rula obligatoriu pe server), JSP page (care preia cererile și generează un conținut dinamic), Java Servlets (cu același rol ca și JSP), JavaBeans (componenta server-side care încapsuleaza comportamente și stări), pagini statice (HTML, XML, DHTML), Java Applets sau JavaBeans (componente client-side și eventual alte fișiere Java de tip class), Java Runtime Environment (care să ruleze pe client și care să poată fi încarcate prin plug-in).

Paginile JSP sunt unice prin faptul că ele pot conține atât conținut, cât și cod de reprezentare. Astfel, se oferă o multitudine de opțiuni în proiectarea de aplicatii ușor de întreținut și extins. Aceste pagini folosesc mai mulți delimitatori pentru funcțiile de scriptare. Cel mai de bază este <% ...%>, care include un script JSP. Un scriptlet este un fragment de cod Java care rulează atunci când utilizatorul solicită pagina.[3]

1. **Descrierea aplicației**

Aplicația are ca scop vizualizarea și gestionarea datelor din baza de date alcătuită din cele 3 tabele: ”mașini”, ”producători” și ”locație\_fabrică” interconectate prin asocierea binară M:1 pentru tabelele ”mașini” și ”producători”, respectiv 1:N pentru ”producători” și ”locație\_fabrică”.

Tabela de asociere ”producatori” realizează legatura dintre tabelele ”mașini” și ”locație\_fabrică”, având ca cheie primară idProducător și ca chei străine cheile primare din aceste tabele, idMașină și idLocație\_Fabrică, realizându-se astfel asocierea N:M între tabelele ”mașini” si ”locație\_fabrică”.

Diagrama bazei de date:

Diagrama bazei de date


Figura 1- Diagrama bazei de date

Diagrama UML a aplicaței[4]

Diagram, schematic

Description automatically generated

Figura 2: Diagrama UML a aplicației

Prima pagină, pagina principală, îi oferă utilizatorului posibilitatea de a alege datele cărei tabele vrea să le vizualizeze dintre cele 3.

Shape, rectangle

Description automatically generated

Figura 3 : Pagina principală

**Tabela Mașini**

**Funcționalitatea de vizualizare**

Dacă se alege tabela Mașini, se vizualizează datele componente ale tabelei din baza de date. Astfel, tabela are pe coloane atributele care se regăsesc și în baza de date.

Table

Description automatically generated

Figura 4: Vizualizare tabela Masini

Pentru realizarea afișării, se va apela funcția vedeTabela din fisierul java JavaBean care primește ca parametru de primire tabela “masini” din baza de date “test2”.

**public** ResultSet vedeTabela(String tabel) **throws** SQLException, Exception {

ResultSet rs = **null**;

**try** {

String queryString = ("select \* from `test2`.`" + tabel + "`;");

Statement stmt = con.createStatement(/\*ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY\*/);

rs = stmt.executeQuery(queryString); /\*executa instructiunea de la functia try si incarca rezultatul in rs-ul pe care il returneaza la final\*/

} **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()

**Funcționalitatea de adăugare**

Prin apăsarea butonului “Adaugă o nouă mașină”, se va deschide o pagină nouă care conține câmpuri libere în vederea completării pentru realizarea adăugării unei noi mașini în baza de date. În urma completării acestora, odată cu apăsarea butonului de final “Adaugă mașina”, se afișează mesajul de confirmare a efectuării operațiunii.

Graphical user interface, application

Description automatically generated

Figura 5 Adăugare în tabela Mașini

Shape, Word

Description automatically generated

Figura 6

Table

Description automatically generated

Figura 7

Se observă că datele adăugate sunt afișate corect în tabela Mașini. Adăugarea unui nou element în această tabelă se realizează în cadrul fișierului nou\_Masina.jsp în care se apelează funcția adaugaMasina() din fișierul JavaBean, funcție ce are ca parametrii atributele tabelei.

**public** **void** adaugaMasina( String Marca, String Model, **int** AnulFabrcatiei, **int** Pret, String Combustibil) **throws** SQLException, Exception {

**if** (con != **null**) {

**try** {

// create a prepared SQL statement

Statement stmt;

stmt = con.createStatement();

stmt.executeUpdate("insert into masini(Marca, Model, AnulFabricatiei, Pret, Combustibil) values('" + Marca + "', '" + Model + "', '" + AnulFabrcatiei + "', '" + Pret + "', '" + Combustibil + "');");

}

**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 adaugaMasina()

**Funcționalitatea de modificare**

Apăsarea butonului butonului “Modifică mașina” conduce spre o nouă pagină care oferă posibilitatea utilizatorului de a selecta ce mașină dorește să modifice.



Figura 8 Modificare tabelă Mașini

Am selectat pentru modificare ultima mașină adăugată. După apăsarea butonului “Modifică linia”, se deschide o nouă fereastră unde se vor nota modificările dorite. Am schimbat anul fabricației mașinii din 2010 în 2013 și am apăsat butonul “Modifică linia”.

Graphical user interface, application

Description automatically generated

Figura 9

Graphical user interface, text, application

Description automatically generated

Figura 10

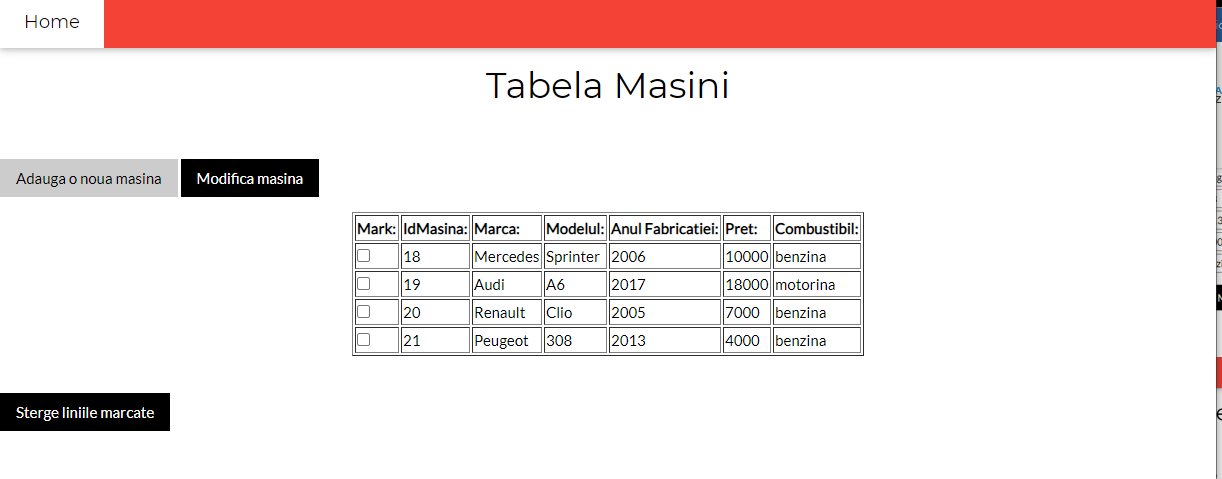


Figura 11

În urma modificării datelor, se afișează mesajul de confirmare a realizării într-o fereastră nouă și în tabela Mașini se observă că datele au fost actualizate cu cele introduse anterior. Pentru realizarea modificării datelor din baza de date, se apelează funcția modificaTabela() din fișierul JavaBean. În cazul realizării acestei funcționalități, în proiectul din Eclipse am creat 3 fișiere cu extensia .jsp : modifica\_Masina.jsp în cadrul căruia s-a făcut vizualizarea tabelei și posbilitatea bifării unei linii ce se dorește a fi modificată, m1\_Masina.jsp în cadrul căruia apar câmpurile ce pot fi modificate de utilizator și m2\_Masina.jsp care afișează mesajul de efectuare cu succes a operațiunii de modificare.

**public** **void** modificaTabela(String tabela, String IDTabela, **int** 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()

**Funcționalitatea de ștergere**

Dacă se dorește ștergerea unui element al tabelei Mașini, se poate selecta linia corespunzătoare acestuia urmată de apăsarea butonului “Șterge liniile marcate”. Am selectat spre exemplificare ultima mașină din tabelă. În urma apăsării butonului, se deschide o pagină nouă cu mesajul de confirmare a operațiunii făcute.

Graphical user interface, table

Description automatically generatedFigura 12 Ștergere linie tabelă Mașini

Graphical user interface, text, application

Description automatically generated

Figura 13

Table

Description automatically generated

Figura 14

Se observă că mașina cu id-ul 21 a fost ștearsă din tabelă. Pentru realizarea acestui lucru, în fișierul sterge\_Masina.jsp creat cu acest scop se apelează funcția stergeDateTabela() din fișierul JavaBean.

**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()

**Tabela Locație\_Fabrică**

**Funcționalitatea de vizualizare**

Dacă în pagina principală se alege tabela Locație Fabrică, se vizualizează datele componente ale tabelei din baza de date. Astfel, tabela are pe coloane atributele care se regăsesc și în baza de date.

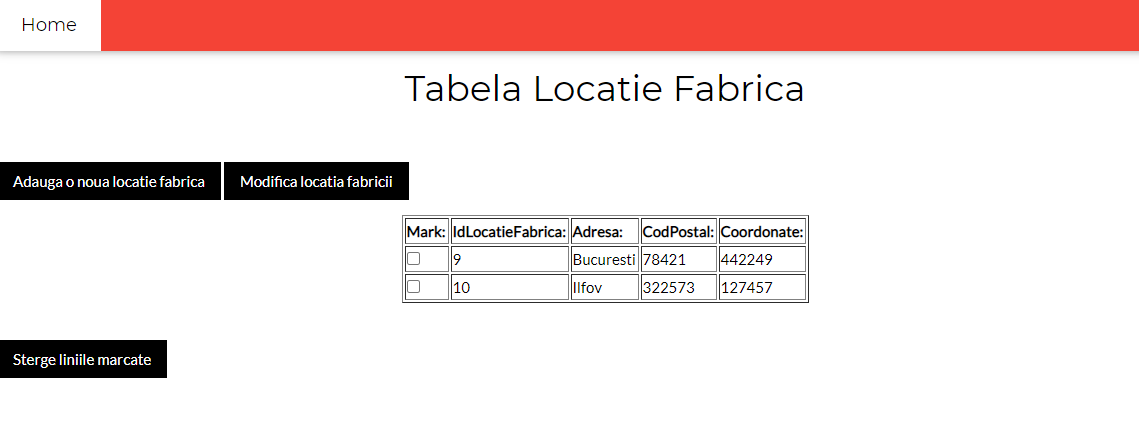


Figura 15 Vizualizare tabela Locație Fabrică

Pentru realizarea afișării, se va apela funcția vedeTabela din fisierul java JavaBean care primește ca parametru de primire tabela “locatie\_fabrica” din baza de date “test2”.

**public** ResultSet vedeTabela(String tabel) **throws** SQLException, Exception {

ResultSet rs = **null**;

**try** {

String queryString = ("select \* from `test2`.`" + tabel + "`;");

Statement stmt = con.createStatement(/\*ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY\*/);

rs = stmt.executeQuery(queryString); /\*executa instructiunea de la functia try si incarca rezultatul in rs-ul pe care il returneaza la final\*/

} **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()

**Funcționalitatea de adăugare**

Prin apăsarea butonului “Adaugă o nouă locație”, se va deschide o pagină nouă care conține câmpuri libere în vederea completării pentru realizarea adăugării unei noi locații în baza de date. În urma completării acestora, odată cu apăsarea butonului de final “Adaugă locația fabricii”, se afișează mesajul de confirmare a efectuării operațiunii.

Graphical user interface, application

Description automatically generated

Figura 16 Adăugare în tabela Locație Fabrică

Shape, Word

Description automatically generated

Figura 17

Graphical user interface, table

Description automatically generated

Figura 18

Se observă că datele adăugate sunt afișate corect în tabela Locație Fabrică. Adăugarea unui nou element în această tabelă se realizează în cadrul fișierului nou\_Locatie\_Fabrica.jsp în care se apelează funcția adaugaLocatie\_Fabrica() din fișierul JavaBean, funcție ce are ca parametrii atributele tabelei.

**public** **void** adaugaLocatie\_Fabrica( String Adresa, **int** CodPostal, **int** Coordonate)

**throws** SQLException, Exception {

**if** (con != **null**) {

**try** {

// create a prepared SQL statement

Statement stmt;

stmt = con.createStatement();

stmt.executeUpdate("insert into locatie\_fabrica(Adresa, CodPostal, Coordonate) values( '" + Adresa + "', '" + CodPostal + "', '" + Coordonate + "');");

} **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 adaugaLocatie\_Fabrica()

**Funcționalitatea de modificare**

Apăsarea butonului butonului “Modifică locație fabrică” conduce spre o nouă pagină care oferă posibilitatea utilizatorului de a selecta ce locație dorește să modifice.

Graphical user interface

Description automatically generated

Figura 19 Modificare element tabelă Locație Fabrică

Am selectat pentru modificare ultima locație adăugată. După apăsarea butonului “Modifică linia”, se deschide o nouă fereastră unde se vor nota modificările dorite. Am schimbat adresa locației din Constanța în Constanța Sud și am apăsat butonul “Modifică linia”.

Graphical user interface, text, application

Description automatically generated

Figura 20

Graphical user interface, text, application

Description automatically generated

Figura 21

Table

Description automatically generated

Figura 22

În urma modificării datelor, se afișează mesajul de confirmare a realizării într-o fereastră nouă și în tabela Locație Fabrică se observă că datele au fost actualizate cu cele introduse anterior. Pentru realizarea modificării datelor din baza de date, se apelează funcția modificaTabela() din fișierul JavaBean. În cazul realizării acestei funcționalități, în proiectul din Eclipse am creat 3 fișiere cu extensia .jsp : modifica\_Locatie\_Fabrica.jsp în cadrul căruia s-a făcut vizualizarea tabelei și posibilitatea bifării unei linii ce se dorește a fi modificată, m1\_Locatie\_Fabrica.jsp în cadrul căruia apar câmpurile ce pot fi modificate de utilizator și m2\_Locatie\_Fabrica.jsp care afișează mesajul de efectuare cu succes a operațiunii de modificare.

**public** **void** modificaTabela(String tabela, String IDTabela, **int** 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()

**Funcționalitatea de ștergere**

Dacă se dorește ștergerea unui element al tabelei Locație Fabrică, se poate selecta linia corespunzătoare acestuia urmată de apăsarea butonului “Șterge liniile marcate”. Am selectat spre exemplificare ultima locație din tabelă. În urma apăsării butonului, se deschide o pagină nouă cu mesajul de confirmare a operațiunii făcute.

Graphical user interface, application

Description automatically generated

Figura 23 Selectare ștergere linie tabelă Locație Fabrică

Graphical user interface, text, application

Description automatically generated

Figura 24

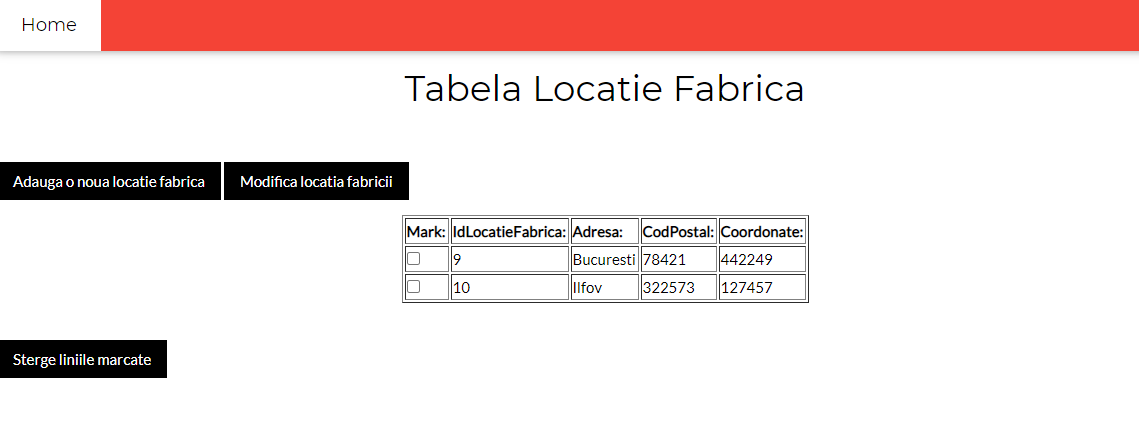


Figura 25

Se observă că locația fabricii cu id-ul 11 a fost ștearsă din tabelă. Pentru realizarea acestui lucru, în fișierul sterge\_Locatie\_Fabrica.jsp creat cu acest scop se apelează funcția stergeDateTabela() din fișierul JavaBean.

**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()

**Tabela Producători**

**Funcționalitatea de vizualizare**

Dacă se alege din pagina principală tabela Producători, se vizualizează datele componente ale tabelei din baza de date. Astfel, tabela are pe coloane atributele care se regăsesc și în baza de date.

Table

Description automatically generated

Figura 26 Vizualizare tabela Producători

Pentru realizarea afișării, se va apela funcția vedeProducator() din fisierul java JavaBean.

**public** ResultSet vedeProducator() **throws** SQLException, Exception {

ResultSet rs = **null**;

**try** {

String queryString = (" select a.Marca MarcaMasinii, a.Model ModelulMasinii, a.AnulFabricatiei AnulFabricatiei, a.Pret PretulMasinii, a.Combustibil Combustibil, b.Adresa AdresaFabricii, b.CodPostal CodPostal, b.Coordonate Coordonate, c.idProducator idProducator, c.idLocatie\_Fabrica idLocatie\_Fabrica, c.idMasina idMasina, c.NumeProducator NumeProducator, c.CifraDeAfaceri CifraDeAfaceri, c.CotaDePiata CotaDePiata from masini a, locatie\_fabrica b, producatori c where a.idMasina = c.idMasina and b.idLocatie\_Fabrica = c.idLocatie\_Fabrica;");

Statement stmt = con.createStatement(/\*ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY\*/);

rs = stmt.executeQuery(queryString);

} **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;

} // vedeProducator()

**Funcționalitatea de adăugare**

Prin apăsarea butonului “Adaugă un nou producător”, se va deschide o pagină nouă care conține câmpuri libere în vederea completării pentru realizarea adăugării unui nou producător, în funcție de mașinile și de locațiile fabricilor care se află în acest moment în baza de date. În urma selectării unei mașini și a locației fabricii unde se află aceasta, se completează cu text restul chenarelor și, odată cu apăsarea butonului de final “Adaugă producătorul”, se afișează mesajul de confirmare a efectuării operațiunii.

Graphical user interface, text, application

Description automatically generated

Figura 27 Adăugare în tabela Producători

Shape, Word

Description automatically generated

Figura 28

Table

Description automatically generated

Figura 29

Se observă că datele adăugate sunt afișate corect în tabela Producători. Adăugarea unui nou element în această tabelă se realizează în cadrul fișierului nou\_Producator.jsp în care se apelează funcția adaugaProducator() din fișierul JavaBean, funcție ce are ca parametrii atributele tabelei.

**public** **void** adaugaProducator( **int** idMasina, **int** idLocatie\_Fabrica, String NumeProducator, **int** CifraDeAfaceri, **int** CotaDePiata) **throws** SQLException, Exception {

**if** (con != **null**) {

**try** {

// create a prepared SQL statement

Statement stmt;

stmt = con.createStatement();

stmt.executeUpdate("insert into producatori(idMasina, idLocatie\_Fabrica, NumeProducator, CifraDeAfaceri, CotaDePiata) values('" + idMasina + "', '" + idLocatie\_Fabrica + "', '" + NumeProducator + "', '" + CifraDeAfaceri + "', '" + CotaDePiata + "');");

}

**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 adaugaProducator()

**Funcționalitatea de ștergere**

Dacă se dorește ștergerea unui element al tabelei Producători, se poate selecta linia corespunzătoare acestuia urmată de apăsarea butonului “Șterge liniile marcate”. Am selectat spre exemplificare ultima linie din tabelă, cea corespunzătoare idProducător = 9. În urma apăsării butonului, se deschide o pagină nouă cu mesajul de confirmare a operațiunii făcute.

Table

Description automatically generated

Figura 30 Selectare linie ștergere tabela Producători

Graphical user interface, text, application

Description automatically generated

Figura 31

Table

Description automatically generated Figura 32

Se observă că producătorul cu id-ul 9 a fost șters din tabelă. Pentru realizarea acestui lucru, în fișierul sterge\_Producator.jsp creat cu acest scop se apelează funcția stergeDateTabela() din fișierul JavaBean.

**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()

**Funcționalitatea de modificare**

Apăsarea butonului butonului “Modifică producator” conduce spre o nouă pagină care oferă posibilitatea utilizatorului de a selecta ce locație dorește să modifice.

Graphical user interface, application

Description automatically generated

Figura 33 Modificare element tabelă Producători

Am selectat pentru modificare utimul producător din tabelă. După apăsarea butonului “Modifică linia”, se deschide o nouă fereastră unde se vor nota modificările dorite. Am schimbat Nume Producător din Audi in BMW și am apăsat butonul “Modifică linia”.

Graphical user interface, application

Description automatically generated

Figura 34

Graphical user interface, text, application

Description automatically generated

Figura 35

Table

Description automatically generated

Figura 36

În urma modificării datelor, se afișează mesajul de confirmare a realizării într-o fereastră nouă și în tabela Producători se observă că datele au fost actualizate cu cele introduse anterior. Pentru realizarea modificării datelor din baza de date, se apelează funcția modificaTabela() din fișierul JavaBean. În cazul realizării acestei funcționalități, în proiectul din Eclipse am creat 3 fișiere cu extensia .jsp : modifica\_Producator.jsp în cadrul căruia s-a făcut vizualizarea tabelei și posibilitatea bifării unei linii ce se dorește a fi modificată, m1\_Producator.jsp în cadrul căruia apar câmpurile ce pot fi modificate de utilizator și m2\_Producator.jsp care afișează mesajul de efectuare cu succes a operațiunii de modificare.

**public** **void** modificaTabela(String tabela, String IDTabela, **int** 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()

**Partea de vizualizare a paginilor:** realizată după un template[5] ; paginile se pot accesa secvențial și se poate ajunge oricând la cea principală (corespunzătoare index-ului) odată cu apăsarea butonului ”Home” care face parte din bara de sus a fiecărei pagini.

**Bibliografie**

[1]: <https://en.wikipedia.org/wiki/MySQL_Workbench>

[2] : <https://ro.scribd.com/doc/147179302/Tehnologia-JSP>

[3] : <https://en.wikipedia.org/wiki/Jakarta_Server_Pages>

[4]: <https://www.lucidchart.com/pages/uml-class-diagram>

[5]: <https://www.w3schools.com/>