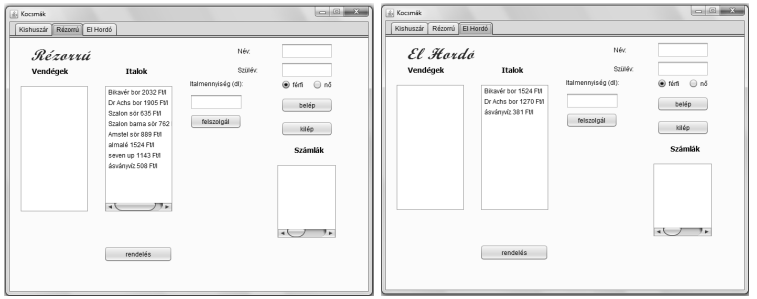
Programozás III

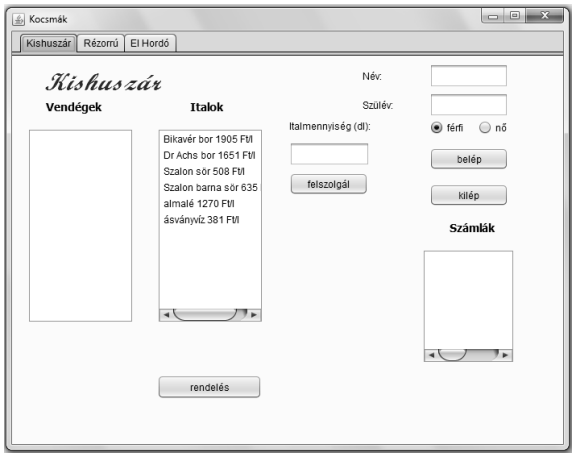
JDBC

PROBLÉMAFELVETÉS

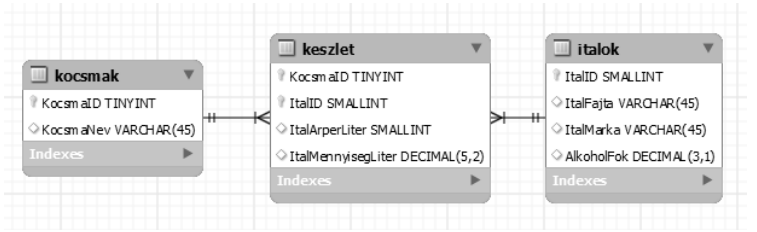


Az adatok adatbázisban vannak.

PROBLÉMAFELVETÉS



PROBLÉMAFELVETÉS



PROBLÉMAFELVETÉS

KocmaID	ItalID	ItalArperLiter	ItalMennyisegLiter
1	1	1500	20
1	2	1300	30
1	3	400	50
1	4	500	40
1	6	1000	30
1	8	300	25
2	1	1600	25
2	2	1500	20
2	3	500	40
2	4	600	40
2	5	700	35
2	6	1200	20
2	7	900	25
2	8	400	30
3	1	1200	60
3	2	1000	40
3	8	300	30

KocsmaID	KocsmaNev
1	Kishuszár
2	Rézorrú
3	El Hordó

ItalID	ItalFajta	ItalMarka	AlkoholFok
1	bor	Bikavér	13.0
2	bor	Dr Achs	12.5
3	sör	Szalon	4.5
4	sör	Szalon barna	5.0
5	sör	Amstel	5.0
6	almalé	NULL	NULL
7	seven up	NULL	NULL
8	ásványvíz	NULL	NULL

ADATBÁZIS-ELÉRÉSI MODELLEK

Kétrétegű modell: a program közvetlenül az adatbázis-kezelő rendszerrel kommunikál. Maga az adatbázis akár másik gépen is elhelyezkedhet, mint ahol a program fut, az adatforgalom a hálózaton keresztül folyik.

Ez az ún. kliens-szerver konfiguráció:
az adatbázist tároló gép a szerver,
a programot futtató gép a kliens

JDBC – A JAVA ADATBÁZIS-KEZELÉSE

A Java hálózati lehetőségei miatt megfelel kliens-szerver architektúrájú adatbázis-kezelő programok létrehozására is.

(Kitérő: ugyanezek a tulajdonságok teszik lehetővé, hogy a Java-t könnyen össze lehessen kötni más programnyelvvel.)

Igény: a Java és tetszőleges adatbázis kommunikációja.

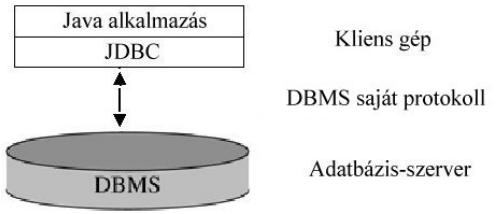
Megoldás: JDBC (Java Database Connectivity)

A JDBC egy Java programozói interfész (API) SQL (Structured Query Language) utasítások végrehajtására.

Segítségével tetszőleges sor-alapú adatforrás kezelhető (akár adatfájlok is).

ADATBÁZIS-ELÉRÉSI MODELLEK

Kétrétegű modell:



(database management system)

ADATBÁZIS-ELÉRÉSI MODELLEK

Háromrétegű modell: a program adott protokollon keresztül egy köztes szolgáltató réteggel kommunikál.

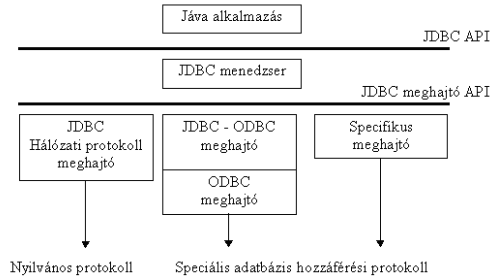
Ez a réteg értelmezi és átalakítja a programtól kapott parancsokat, majd továbbítja őket az adatbázis-kezelő rendszerhez.
A lekérdezési eredményeket is ezen a rétegen keresztül kapja meg a program.

A középső réteg
- lehetővé teszi az adatbázishoz való hozzáférés és tranzakció-kezelés felügyeletét,
- megkönnyíti az alkalmazások továbbfejlesztését,
- hatékonyabb.

Középső réteg régen C, C++ ma már Java

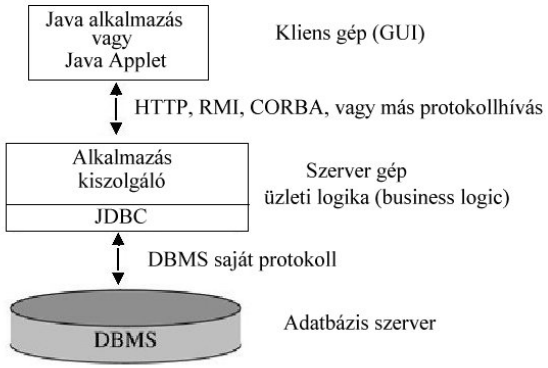
JDBC – MOST CSAK MINIMÁLIS ELMÉLETI HÁTTÉRREL

A JDBC két felületet határoz meg:
A felső felület az a programozói interfész, amely segítségével a programozók elérhetik az adatbázis szolgáltatásait.
Az alsó szint különböző adatbázisok elérését megvalósító meghajtó programok implementációjánál játszik szerepet.



ADATBÁZIS-ELÉRÉSI MODELLEK

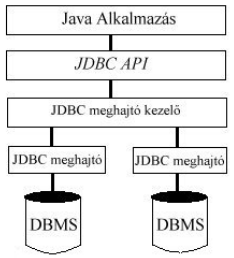
Háromrétegű modell:



A JDBC FELÉPÍTÉSE

A JDBC fő célja egy adatbázis-független interfész, egy „általános SQL adatbázis hozzáférési keretrendszer”.

A programozó egyetlen adatbázis interfészt készít, és a JDBC használatával a program gond nélkül képes hozzáférni bármilyen adatforráshoz.



JDBC MEGHAJTÓ-PROGRAMOK

A JDBC hívások végrehajtásakor mindig fizikailag is fel kell venni a kapcsolatot a felhasznált adatbázissal.

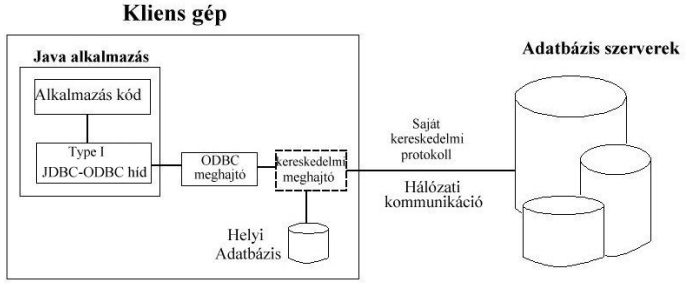
Az adatbázis-kezelő bármilyen (adatbázis) szoftver lehet.

Minden adatbázis-kezelő esetén külön biztosítani kell a JDBC hívások megfelelő értelmezését és kiszolgálását.

Ezt a feladatot végzik el a JDBC meghajtó-programok. (Egy adott adatbázis-típushoz implementálva a Driver interfészt.)

JDBC MEGHAJTÓ-PROGRAM TÍPUSOK

1. JDBC – ODBC áthidaló-program + ODBC (Open Database Connectivity) meghajtó-program (Type I) : JDBC hívások kiszolgálása ODBC közbeiktatásával:



Már létező ODBC programcsomag használatát teszi lehetővé.

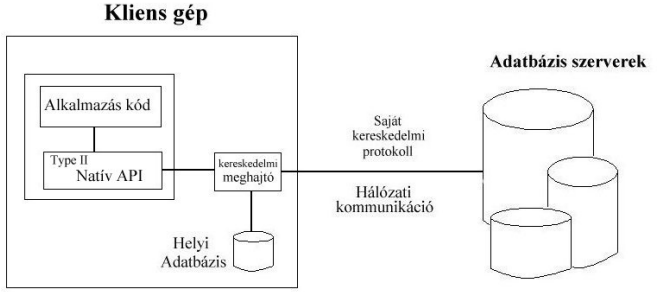
JDBC MEGHAJTÓ-PROGRAMOK

A meghajtónak implementálnia kell mindazokat az osztályokat, amelyek megvalósítják az adatbázis funkcióit. Ezek a merev követelmények biztosítják, hogy a meghajtó minden esetben teljesítse azt, amit elvárunk tőle.

A jelenlegi meghajtó-programoktól már elvárható, hogy többszálúan valósítsanak meg egy kapcsolatot, vagyis azonos kapcsolaton belül kiadott több SQL utasítás feldolgozásának párhuzamosítása már nem a programozó, hanem a meghajtó-program feladata.

JDBC MEGHAJTÓ-PROGRAM TÍPUSOK

2. JDBC – natív kliens-API: (Type II) :
A Java meghajtó-program a JDBC hívásokat közvetlenül átalakítja a megfelelő adatbázis kliens natív API hívásaira. Minden gépen ott kell lennie a megfelelő adatbázis kliens-API-t megvalósító bináris program(könyvtár)-nak.



JDBC MEGHAJTÓ-PROGRAM TÍPUSOK

3. JDBC – hálózati protokoll (Type III) :
A Java-ban írt meghajtóprogram a JDBC hívásokat adatbázis-független protokoll-hívásokká alakítja, amelyeket egy szerverprogram értelmez és lefordít az adott adatbázis-kezelő API-jának hívásaira.

A kliens nem közvetlenül az adatbázissal, hanem egy köztes szerverrel kommunikál, és csak ez a szerver-program tart fenn kapcsolatot az adatbázissal (3-rétegű adatbázis-elérési modell).

API: Application Programming Interface

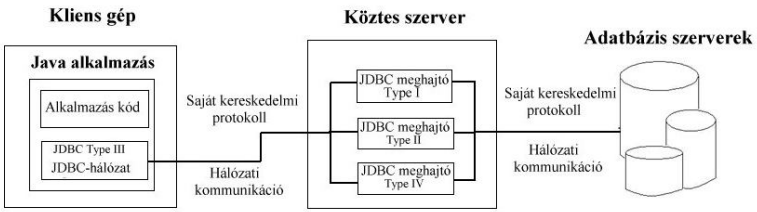
JDBC MEGHAJTÓ-PROGRAM TÍPUSOK

4. JDBC – saját adatbázis protokoll (Type IV) :
Java-ban írt meghajtóprogram, amely a hívásokat közvetlenül az adatbázis-kezelő nyelvére alakítja át. Nincs szükség közbülső szerverprogramra, hiszen a meghajtó-program a hálózaton keresztül közvetlenül az adatbázis-kezelővel kommunikál.

Hátránya viszont, hogy ezek a driver-ek adatbázisfüggők.

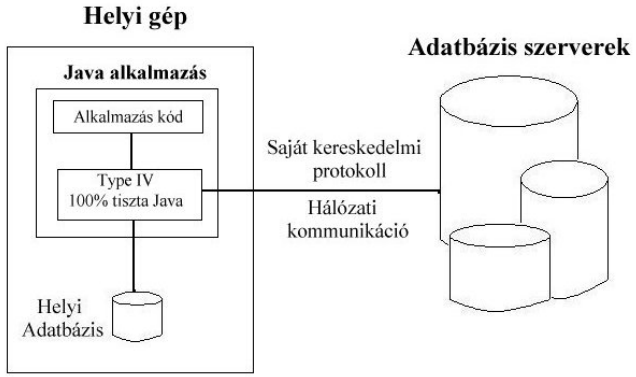
JDBC MEGHAJTÓ-PROGRAM TÍPUSOK

3. JDBC – hálózati protokoll (Type III) :



JDBC MEGHAJTÓ-PROGRAM TÍPUSOK

4. JDBC – saját adatbázis protokoll (Type IV) :



1-2. platformfüggő, 3-4. platformfüggetlen

A MEGHAJTÓ-KEZELŐ

A meghajtó-kezelő (*DriverManager*) felelős a kapcsolat megnyitásáért. Azonban még a kapcsolódás előtt **regisztrálni** kell a megfelelő JDBC meghajtót.

Egy JDBC alkalmazás egy vagy több drivert is használhat. A driverek adatbázis specifikusak, tehát minden adatbázis más meghajtót használ. Ezért ha különböző adatbázisokat használunk, akkor több driver is szükséges. A kapcsolat létrehozásakor a meghajtó-kezelő kiválasztja a regisztrált meghajtók közül azt, amelyik megfelel az aktuális adatbázisnak.

Miután létrejött a kapcsolat, a program közvetlenül a JDBC meghajtón keresztül kommunikál az adatbázissal.

MEGHAJTÓ-PROGRAMOK KEZELÉSE

A meghajtó-program valósítja meg a fizikai kommunikációt a program és a megfelelő adatbázis között.

Eszközei:

- java.sql.**Driver** interfész
  ezt minden meghajtó-programnak implementálnia kell
- java.sql.**DriverManager** osztály
  - feladata a megfelelő meghajtó-osztály kiválasztása és használata
  - nyilvántartja a pillanatnyilag használható összes regisztrált meghajtó-programot
  - aktivizálja a megfelelő meghajtó-programot
  - ellenőrzési, nyomkövetési funkciók

A JDBC HASZNÁLATA

Adatbázis-kezelést alkalmazó Java programok felépítése:

Csomagok:

java.sql

javax.sql

MEGHAJTÓ-PROGRAMOK REGISZTRÁLÁSA

A meghajtó-program akkor válik hozzáférhetővé egy Java programban, ha regisztráljuk (DriverManager osztály metódusai).
Két módszer:

1. a meghajtó-program osztály direkt betöltése

   Class.forName(String *driverName*);

Pl.:
```
String meghajto = ”sun.jdbc.odbc.JdbcOdbcDriver”;
try{
        Class.forName(meghajto);
}
catch (ClassNotFoundException e){
        e.printStackTrace();
}
```

MEGHAJTÓ-PROGRAMOK REGISZTRÁLÁSA

Másik módszer:

a jdbc.drivers rendszerparaméter beállításával

Nem igazán javasolt megoldás, mert :

- csak egyszer, az inicializáláskor veszi figyelembe a beállítást
- appletek esetén nem használható

Megjegyzés:

betölthető driver-eket (ill. egy részét) ld.:

…\jre…\lib\rt.jar – vagy letölthető

KAPCSOLATTARTÁS AZ ADATBÁZISSAL

Adatbázis URL-ek

Az elérni kívánt adatbázis azonosítására szolgál. Itt adjuk meg, hogy melyik adatbázishoz szeretnénk kapcsolódni.

Általános struktúrája:
jdbc:<*al-protokoll*>:<*adatforrás leírása*>

al-protokoll:
a megfelelő meghajtó-programra utal

adatforrás leírása:
meghatározza a kért adatbázis eléréséhez szükséges adatokat.

Pl.: "jdbc:mysql://szerver_cime/adatbázis_neve"

KAPCSOLATTARTÁS AZ ADATBÁZISSAL

A program és az adatbázis között a kapcsolatot a **Connection** osztály reprezentálja.

Tulajdonságai:

- Egy program egyszerre több kapcsolatot is fenntarthat ugyanavval vagy különböző adatbázisokkal.
- Egy kapcsolat a kiadott SQL utasításokat és azok végrehajtásának eredményét foglalja magában.
- Többszálúság, vagyis az azonos kapcsolaton belül kiadott különböző SQL utasítások feldolgozásának párhuzamosítása már nem a programozó, hanem a meghajtó-program feladata.

KAPCSOLATTARTÁS AZ ADATBÁZISSAL

A kapcsolat felvétele

Módja: DriverManager osztály getConnection metódusa

paramétere: a kívánt adatbázis URL címe,
esetleg felhasználónév, jelszó, egyéb tul-k.

Pl.: Connection kapcsolat;

kapcsolat = DriverManager.getConnection(url, user, jelszo);

Működése:
A DriverManager a regisztrálás sorrendjében végignézi, hogy a regisztrált meghajtó-programok közül melyik tud kapcsolódni az URL-ben megadott adatbázishoz, és kapcsolódik, ha tud.

KAPCSOLATTARTÁS AZ ADATBÁZISSAL

A kapcsolat lezárása

close():

felszabadítja az adatbázis-kapcsolat által lefoglalt JDBC erőforrásokat.

Feledékenység esetén:

„hulladékgyűjtés” (Garbage Collector)

SQL UTASÍTÁSOK VÉGREHAJTÁSA

A Statement interfész

Statement objektum létrehozása:
egy fennálló kapcsolatot reprezentáló Connection objektum createStatement metódusával.

Pl.:

```
Connection kapcsolat =
        DriverManager.getConnection( url, user, jelszo);
Statement utasitasObjektum = kapcsolat.createStatement();
```

SQL UTASÍTÁSOK VÉGREHAJTÁSA

A JDBC adatbázis-független ↓

Kezelnie kell az adatbázis-kezelő rendszerek SQL megvalósításai közötti apró eltéréseket.

Használható interfészek:

Statement: egyszerű SQL utasítások végrehajtására.

PreparedStatement: bemenő paraméterekkel rendelkező SQL utasítások végrehajtására.

CallableStatement: ki/bemenő paraméterekkel is rendelkező tárolt SQL eljárások végrehajtására

SQL UTASÍTÁSOK VÉGREHAJTÁSA

Statement végrehajtása

– executeQuery
végrehajtja a paraméterében megadott SQL utasítást, és visszaadja az eredményt reprezentáló eredménytáblát.

Végrehajtható utasítás: SELECT
Eredmény: egy ResultSet objektum.

SQL UTASÍTÁSOK VÉGREHAJTÁSA

Statement végrehajtása (folyt.)

– executeUpdate

Adatmanipulációs (UPDATE, INSERT, DELETE) és adatdefiníciós (CREATE TABLE, DROP TABLE…stb) SQL utasítások végrehajtására használható.

Eredményül a megváltozott adatbázistábla-sorok számát adja vissza.

– execute

Olyan esetekben használandó, ha az utasítás többfajta eredményt is visszaadhat (pl. tárolt eljárások) vagy nem ismert, hogy milyen eredményt ad vissza.

EREDMÉNYEK FELDOLGOZÁSA

Navigálás az eredménytáblában

Az eredménytáblának mindig csak az aktuális sora érhető el. Ezt a sort egy ún. SQL kurzor jelöli, amely kezdetben az eredménytábla első sora előtt áll. (Ezért kiolvasáshoz mindig kell a next().)

Néhány navigáló metódus: next

previous

last

first

stb…

EREDMÉNYEK FELDOLGOZÁSA

Az SQL eredmények feldolgozása: ResultSet objektum.

A ResultSet objektum néhány tulajdonsága:

– ez egy logikai nézet, mely az adatbázis sorainak és oszlopainak az SQL lekérdezés szerinti megfeleltetése;

– bármennyi sorból és oszlopból állhat;

– lehetőséget ad arra, hogy megtekintsük a lekérdezés feltételeinek megfelelő adatokat;

– lehetőség van adatok frissítésére, illetve új sor beszúrására vagy törlésére is.

EREDMÉNYEK FELDOLGOZÁSA

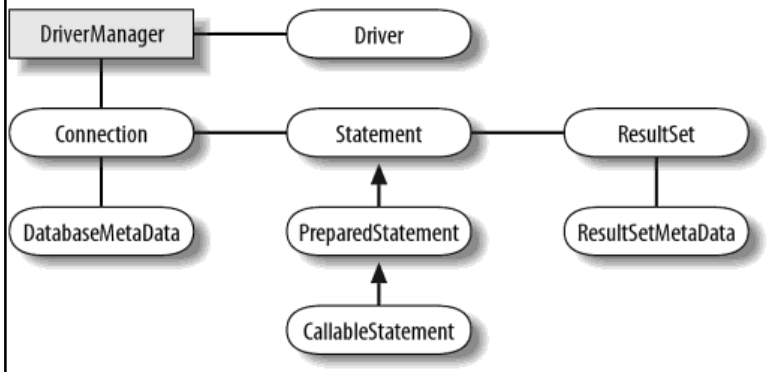
Adatok kinyerése az eredménytáblából

Az SQL lekérdezés eredményeként kapott adatok JDBC adattípusok. Ahhoz, hogy programunkban használni tudjuk, először át kell alakítani őket Java adattípusokká.

A ResultSet objektum metódusai biztosítják a konverziót, pl.:

```
ResultSet eredmenyHalmaz;
eredmenyHalmaz.getString(”oszlopnev”);
eredmenyHalmaz.getTime(”oszlopnev”);
eredmenyHalmaz.getInt(”oszlopnev”);
stb…
```

ÖSSZEFOGLALÓ ÁBRA



JDBC HASZNÁLATA APPLETEKBEN

Appletek használatakor a felhasználó a JDBC-n keresztül hozzáférhet az adatbázishoz.

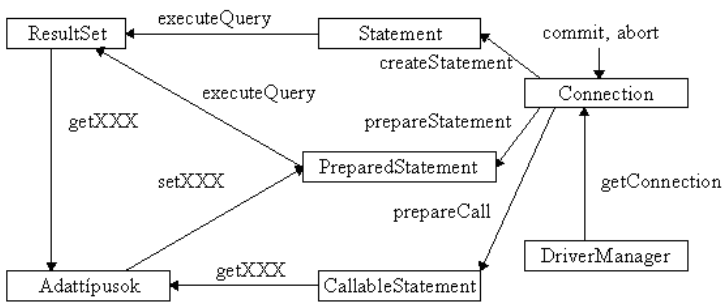
Ennek feltétele:

A kliens-oldalról elérhető legyen a használt adatbázis-meghajtó program kódja.

Egy applet csak a kódját tartalmazó géppel hozhat létre hálózati kapcsolatot.

Csak olyan adatbázissal kommunikálhat, amely ugyanazon a szerveren fut, mint ahonnan az applet kódja is letöltésre került. ☹

ÖSSZEFOGLALÓ ÁBRA



JDBC HASZNÁLATA APPLETEKBEN

Megoldás:

Többrétegű elérési modell.

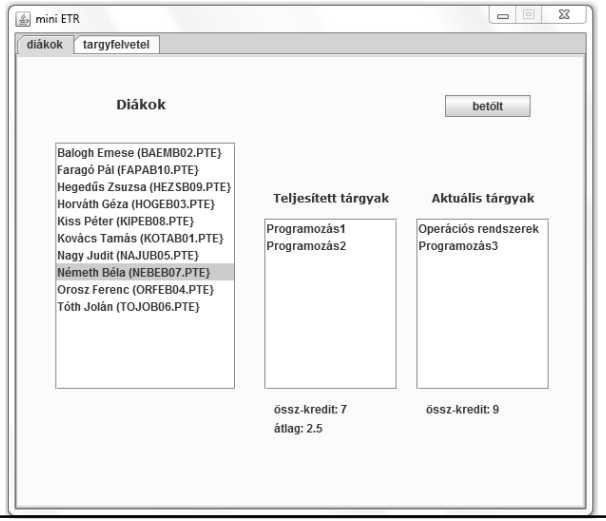
Ekkor csak a kiszolgáló rétegnek kell ugyanazon a gépen lennie, mint az applet kódjának, maga az adatbázis bárhol máshol is lehet, az applet el tudja érni a kiszolgáló rétegen keresztül.

EGYSZERŰ ALKALMAZÁSI PÉLDÁK

1. feladat: MySQL adatbázis feldolgozása
2. feladat: Derby használata
3. feladat: MsAccess példa

EGYSZERŰ ALKALMAZÁSI PÉLDÁK

1. Feladat:

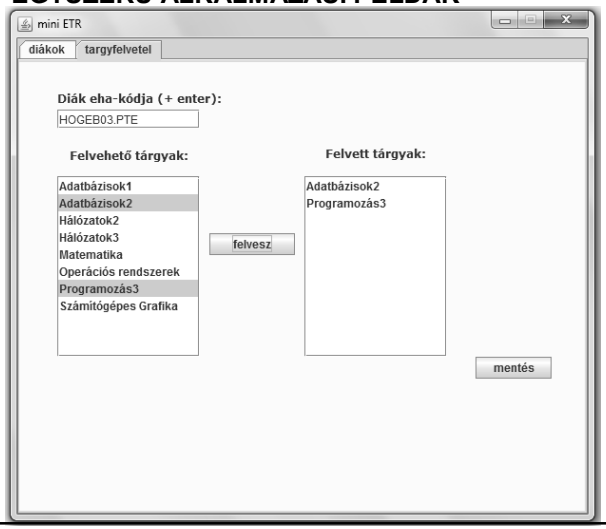


EGYSZERŰ ALKALMAZÁSI PÉLDÁK

1. Feladat:

EGYSZERŰ ALKALMAZÁSI PÉLDÁK

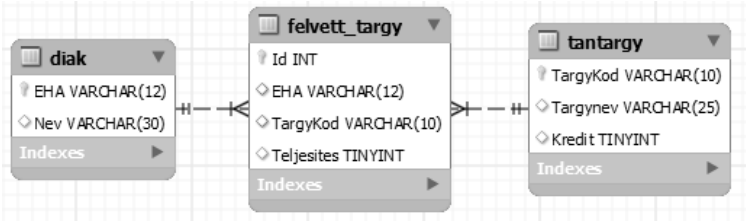
1. Feladat:



Az 1. FELADAT MEGOLDÁSA

Megoldás:

Most egy alábbi szerkezetű MySQL adatbázist használunk:



Az 1. FELADAT MEGOLDÁSA

Elvileg bármilyen adatbáziskezelő használható.
Szükség van még az adatbázishoz tartozó JDBC driverre, és lehetővé kell tennünk, hogy ez a projektből is elérhető legyen. (Libraries/Add Jar)

Jelen esetben MySQL adatbázist használunk. A szükséges driver:

mysql-connector-java-5.1.36-bin.jar

(letölthető az internetről)

Az 1. FELADAT MEGOLDÁSA

Létre kell hoznunk az adatbázist, és feltölteni adatokkal.

EHA	Nev
BAEMB02.PTE	Balogh Emese
FAPAB10.PTE	Faragó Pál
HEZSB09.PTE	Hegedűs Zsuzsa
HOGEB03.PTE	Horváth Géza
KIPEB08.PTE	Kiss Péter
KOTAB01.PTE	Kovács Tamás
NAJUB05.PTE	Nagy Judit
NEBEB07.PTE	Németh Béla
ORFEB04.PTE	Orosz Ferenc
TOJOB06.PTE	Tóth Jolán

Id	EHA	TargyKod	Teljesites
1	BAEMB02.PTE	AB01	0
2	BAEMB02.PTE	HALO01	0
3	BAEMB02.PTE	MAT01	0
4	BAEMB02.PTE	PRG01	3
5	FAPAB10.PTE	AB01	0
6	FAPAB10.PTE	HALO01	0
7	FAPAB10.PTE	MAT01	0
8	HOGEB03.P...	HALO01	0
9	HOGEB03.P...	PRG01	4
10	HOGEB03.P...	PRG02	0
11	KIPEB08.PTE	MAT01	0
12	KOTAB01.PTE	MAT01	0
13	KOTAB01.PTE	PRG01	0
14	NEBEB07.PTE	OPR01	0
15	NEBEB07.PTE	PRG01	2
16	NEBEB07.PTE	PRG02	3
17	NEBEB07.PTE	PRG03	0
25	NAJUB05.PTE	AB01	0
26	NAJUB05.PTE	MAT01	0

TargyKod	Targynev	Kredit
AB01	Adatbázisok1	4
AB02	Adatbázisok2	3
HALO01	Hálózatok1	3
HALO02	Hálózatok2	3
HALO03	Hálózatok3	3
MAT01	Matematika	4
OPR01	Operációs rendszerek	4
PRG01	Programozás1	3
PRG02	Programozás2	4
PRG03	Programozás3	5
SZGR01	Számítógépes Grafika	2

Az 1. FELADAT MEGOLDÁSA

A projekt szerkezete:

Az 1. FELADAT MEGOLDÁSA

```
public class ETRFrame extends javax.swing.JFrame {

    private InditoPanel inditas = new InditoPanel();
    private JdbcPanel jdbcPanel;
    private AdatBazis adatBazis = AdatBazis.getPeldany();

    private int szelesseg = 600, magassag = 500;
    private String cim = "mini ETR";

    public ETRFrame() {
        initComponents();
        this.setSize(szelesseg, magassag);
        this.setTitle(cim);
        this.setLocationRelativeTo(null);
    }
                              public void run() {
                                  ETRFrame frame = new ETRFrame();
                                  frame.setVisible(true);
                                  frame.beallitasok();
                              }
```

Az 1. FELADAT MEGOLDÁSA

```
    private Connection kapcsolat;
    private Statement allitas;
    private ResultSet eredmeny;

public void kapcsolodas() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://szerver/miniETR";
        kapcsolat = DriverManager.getConnection(url, "user", "jelszo");

    } catch (ClassNotFoundException | SQLException ex) {
        Logger.getLogger(AdatBazis.class.getName()).log(Level.SEVERE, x
    }
}
```

Meghajtó-program betöltése

Adatbázis URL

7-es JDK-tól

Az 1. FELADAT MEGOLDÁSA

```
private void beallitasok() {
    this.getContentPane().add(inditas,BorderLayout.CENTER);
    inditas.setFrame(this);

    adatBazis.kapcsolodas();

    this.addWindowListener(new WindowAdapter(){
        @Override
        public void windowClosing(WindowEvent e) {
            adatBazis.lezar();
        }
    });
}
```

Az 1. FELADAT MEGOLDÁSA

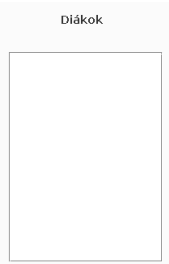
```
public void lezar(){
    try {
        kapcsolat.close();
        System.out.println("lezárva");
    } catch (SQLException ex) {
        Logger.getLogger(AdatBazis.class.getName()).log(Level.SEVERE,
    }
}
```

Az 1. FELADAT MEGOLDÁSA

Az induló panel váltása:

```
    private void ikonLabelMouseClicked(java.awt.event.MouseEvent evt) {
        frame.valtas();
}

  void valtas() {
      this.getContentPane().remove(inditas);
      jdbcPanel = new JdbcPanel(this);
      jdbcPanel.setAdatBazis(adatBazis);
      this.getContentPane().add(jdbcPanel, BorderLayout.CENTER);
      this.pack();
  }
```

Az 1. FELADAT MEGOLDÁSA




```
    private void betoltGombActionPerformed(java.awt.event.ActionEvent evt) {
        diakModell.clear();
        List<Diak> diakok = adatBazis.betoltes();
        for(Diak diak: diakok){
            diakModell.addElement(diak);
        }
}
```

Az 1. FELADAT MEGOLDÁSA

Beállítások:

```
public class JdbcPanel extends javax.swing.JPanel {

    private DefaultListModel<Diak> diakModell = new DefaultListModel<>();
    private DefaultListModel<String> teljesitettTargyModell = new DefaultListModel<>();
    private DefaultListModel<String> aktualisTargyModell = new DefaultListModel<>();
    private DefaultListModel<String> felvehetoTargyModell = new DefaultListModel<>();
    private DefaultListModel<String> felvettTargyModell = new DefaultListModel<>();
    private String eha;
    private AdatBazis adatBazis;

          public JdbcPanel(ETRFrame frame) {
              initComponents();
              diakLista.setModel(diakModell);
              teljesitettLista.setModel(teljesitettTargyModell);
              aktualisLista.setModel(aktualisTargyModell);
              felvehetoLista.setModel(felvehetoTargyModell);
              felvettLista.setModel(felvettTargyModell);
              this.setSize(frame.getWidth(), frame.getHeight());
              mentGomb.setEnabled(false);
          }
```

Az 1. FELADAT MEGOLDÁSA

```
public List<Diak> betoltes() {
    List <Diak> diakok = new ArrayList<>();
    String nev, eha;
    try {
        if (kapcsolat != null) {
            allitas = kapcsolat.createStatement();
            eredmeny = allitas.executeQuery("SELECT * FROM diak ;");
            while (eredmeny.next()) {
                nev = eredmeny.getString("Nev");
                eha =  eredmeny.getString("EHA");
                diakok.add(new Diak(nev, eha, this));
            }
        }
    } catch (SQLException ex) {
        Logger.getLogger(AdatBazis.class.getName()).log(Level.SEVERE,
    }
    return diakok;
}
```

```java
public class Diak {

    private String nev, eha;
    private AdatBazis adatBazis;

    public Diak(String nev, String eha, AdatBazis adatBazis) {
        this.nev = nev;
        this.eha = eha;
        this.adatBazis = adatBazis;
    }

    public float teljesitettAtlag(){
        return adatBazis.teljesitettAtlag(eha);
    }

    public int aktualisKredit(){
        return adatBazis.aktualisKredit(eha);
    }

    public int teljesitettKredit(){
        return adatBazis.teljesitettKredit(eha);
    }

    @Override
    public String toString() {
        return nev + " (" + eha + ")" ;
    }
```

+ get metódusok

```java
    private void diakListaValueChanged(javax.swing.event.ListSelectionEvent evt) {
        Diak valasztott = (Diak) diakLista.getSelectedValue();
        eha = valasztott.getEha();
        teljesitettTargyModell.clear(); aktualisTargyModell.clear();
        List<String> teljesitettek = adatBazis.teljesitettTargyak(eha),
                     aktualisok = adatBazis.aktualisTargyak(eha);
        for (String targy : teljesitettek) {
            teljesitettTargyModell.addElement(targy);
        }
        for (String targy : aktualisok) {
            aktualisTargyModell.addElement(targy);
        }
        if (teljesitettLista.getLastVisibleIndex() >= 0) {
            teljesitettKreditLabel.setText("össz-kredit: "
                                           + valasztott.teljesitettKredit());
            atlagLabel.setText("átlag: " + valasztott.teljesitettAtlag());
        } else {
            atlagLabel.setText("");
            teljesitettKreditLabel.setText("");
        }
        if (aktualisLista.getLastVisibleIndex() >= 0) {
            aktualisKreditLabel.setText("össz-kredit: "
                                        + valasztott.aktualisKredit());

        } else {
            aktualisKreditLabel.setText("");
        }
}
```

Az 1. FELADAT MEGOLDÁSA

egyszeres kiválaszthatóság

Az 1. FELADAT MEGOLDÁSA

```java
public List<String> teljesitettTargyak(String eha){
    List<String> targyak = new ArrayList<>();
    String sqlParancs;
    try {
        if (kapcsolat != null) {
            allitas = kapcsolat.createStatement();
            sqlParancs = " SELECT TargyNev FROM tantargy INNER JOIN felvett_targy "
                       + " ON tantargy.TargyKod = felvett_targy.TargyKod "
                       + " WHERE felvett_targy.EHA = '"+eha+"' "
                       + " AND felvett_targy.teljesites > 1; " ;
            eredmeny = allitas.executeQuery(sqlParancs);

            while (eredmeny.next()) {
                targyak.add(eredmeny.getString("TargyNev") );
            }
        }
    } catch (SQLException ex) {
        Logger.getLogger(AdatBazis.class.getName()).log(Level.SEVERE, null, ex);
    }
    return targyak;
}
```

Az 1. FELADAT MEGOLDÁSA

```
/* Alternatívák:
   sqlParancs = " SELECT TargyNev FROM tantargy INNER JOIN felvett_targy "
              + " ON tantargy.TargyKod = felvett_targy.TargyKod "
              + " WHERE felvett_targy.EHA = '"+eha+"' "
              + " AND felvett_targy.teljesites > 1; " ;
   eredmeny = keres.executeQuery(sqlParancs); */

/* sqlParancs = " SELECT TargyNev FROM tantargy "
              + " WHERE TargyKod "
              + " IN (SELECT TargyKod FROM felvett_targy "
              + "     WHERE felvett_targy.EHA = '"+eha+"' "
              + "     AND felvett_targy.teljesites > 1); " ; */
```

Az aktuális tárgyak kiválasztása ugyanilyen, csak a teljesítés még 0.

Az 1. FELADAT MEGOLDÁSA

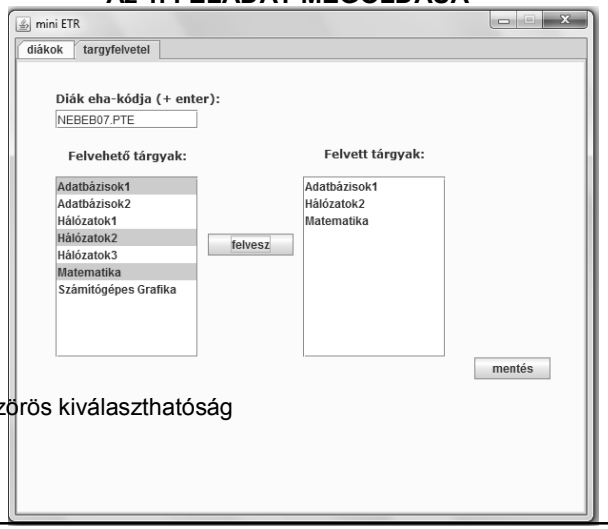
A teljesített kreditek számának meghatározására egy lehetséges lekérdezés (de a korábban említett alternatívák alapján is lehet) – és hasonló az aktuális kreditek számának meghatározása is.

```
sqlParancs = " SELECT sum(Kredit) as kredit FROM tantargy "
           + "                          INNER JOIN felvett_targy "
           + " ON tantargy.TargyKod = felvett_targy.TargyKod "
           + " WHERE felvett_targy.EHA = '"+eha+"' "
           + " AND felvett_targy.teljesites > 1; " ;
```

Az 1. FELADAT MEGOLDÁSA

```
public float teljesitettAtlag(String eha) {
     float atlag=0;
     String sqlParancs;
     try {
         if (kapcsolat != null) {
             allitas = kapcsolat.createStatement();
             sqlParancs = " SELECT avg(Teljesites) as atl FROM felvett_targy "
                        + " WHERE felvett_targy.EHA = '"+eha+"' "
                        + " AND felvett_targy.Teljesites > 1; " ;
             eredmeny = allitas.executeQuery(sqlParancs);

             if (eredmeny.next()) {
                    atlag = eredmeny.getFloat("atl");
             }
         }
     } catch (SQLException ex) {
         Logger.getLogger(AdatBazis.class.getName()).log(Level.SEVERE, null, ex
     }
     return atlag;
 }
```

Az 1. FELADAT MEGOLDÁSA



többszörös kiválaszthatóság

Az 1. FELADAT MEGOLDÁSA

```
    private void ehaMezoKeyPressed(java.awt.event.KeyEvent evt) {
        if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
            eha = ehaMezo.getText();
            felvettTargyModell.clear();
            for(String targy : adatBazis.felvehetoTargyak(eha)){
                felvehetoTargyModell.addElement(targy);
            }
        }
}
    private void felveszGombActionPerformed(java.awt.event.ActionEvent evt) {
        boolean valtozott = false;
        List<String> felvehetok = felvehetoLista.getSelectedValuesList();
        for(String targy : felvehetok){
            if(!felvettTargyModell.contains(targy)){
                felvettTargyModell.addElement(targy);
                valtozott = true;
            }
        }
        if(valtozott)mentGomb.setEnabled(true);
}
```

Az 1. FELADAT MEGOLDÁSA

```
public void felvettTargyMentes(String eha, List<String> felvettTargyak) {
    String sqlParancs;
    try {
        if (kapcsolat != null) {
            allitas = kapcsolat.createStatement();
            for (String targy : felvettTargyak) {
                sqlParancs = " INSERT INTO felvett_targy (EHA, TargyKod, Teljesites) "
                           + " VALUES('" + eha + "',"
                           + "         (SELECT TargyKod FROM tantargy "
                           + "          WHERE tantargy.TargyNev = '" + targy + "'), "
                           + "         0 )" ;
                allitas.executeUpdate(sqlParancs);
            }
        }
    } catch (SQLException ex) {
        Logger.getLogger(AdatBazis.class.getName()).log(Level.SEVERE, null, ex);
    }
}
     return atlag;
  }
```

Az 1. FELADAT MEGOLDÁSA

```
    private void mentGombActionPerformed(java.awt.event.ActionEvent evt) {
        List<String> felvettek = new ArrayList<>();
        for(int i=0; i<felvettTargyModell.size(); i++){
            felvettek.add(felvettTargyModell.get(i));
        }
        adatBazis.felvettTargyMentes(eha, felvettek);
        mentGomb.setEnabled(false);
}
```

Az 1. FELADAT MEGOLDÁSA

Feladat:

A BEMUTATOTT PÉLDA ÖNÁLLÓ KIPRÓBÁLÁSA!

IRODALOM

JDBC - MYSQL

http://www.developer.com/java/data/article.php/3417381/Using-JDBC-with-MySQL-Getting-Started.htm#Discussion%20and%20Sample%20Programs

http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

+ google

2. PÉLDA

Néhány jónak tűnő kiindulási alap:

https://netbeans.org/kb/docs/ide/java-db.html

http://www.youtube.com/watch?v=aaAk8J2OcQA (kicsit szájbarágós)

http://db.apache.org/derby/papers/DerbyTut/embedded_intro.html

http://db.apache.org/derby/docs/10.9/getstart/

http://www.zetcode.com/db/apachederbytutorial/

+ google

2. PÉLDA

Feladat:
Ismerkedjen meg a Derby adatbázis-kezelő rendszerrel!

Apache Derby:

- Java nyelven implementált nyílt forrású relációs adatbázis-kezelő.
- JDBC, SQL kezelés
- Lehet:
  - Beágyazott adatbázis
  - Kliens – szerver típusú adatbáziskezelés

2. PÉLDA

Néhány megjegyzés a https://netbeans.org/kb/docs/ide/java-db.html tutoriálhoz:

- ne felejtsen el csatlakozni az adatbázishoz

- a tábla létrehozásakor nem kell bekapcsolni a chech opciót - ha valami miatt nem fogadja el, hozzon létre egy egyetlen oszlopból álló táblát, aztán ehhez adja egyenként az oszlopokat, és kísérletezze ki, mi okozhatja a bajt.

2. PÉLDA

A tutoriálban megadott adatbázis elérése – részlet:

```
public void adatbazsiKapcsolat() throws Exception {
    // kliens alapú
    // szükséges driver: derbyclient.jar
//  Class.forName("org.apache.derby.jdbc.ClientDriver");
//  String url = "jdbc:derby://localhost:1527/contact";
//  kapcsolat = DriverManager.getConnection(url, "nbuser", "jelszo");

    // beágyazott - a ContactDB helye: src mellett
    // szükséges driver: derby.jar
    Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
    String url = "jdbc:derby:ContactDB";
    kapcsolat = DriverManager.getConnection(url, "nbuser", "jelszo");
    System.out.println("létrejött a kapcsolat");

    if(kapcsolat != null){
        Statement allitas = kapcsolat.createStatement();
        ResultSet eredmeny = allitas.executeQuery("SELECT * FROM APP.FRIENDS");
        while(eredmeny.next()){
            System.out.println(eredmeny.getString("FIRSTNAME"));
        }
    }
}
```

3. PÉLDA

Feladat:

Hozzunk létre egy MsAccess adatbázist, és írassuk ki az elemeit egy Java programban.

(Ha van egy kis rutinja az adatbázis-kezelésben, akkor inkább az előző két megoldástípus valamelyikét próbálja ki.)

3. PÉLDA

Megoldás:

Szükségünk van egy meghajtóra, amely megérti a JDBC parancsokat, lefordítja és továbbítja őket az adatbázishoz.

A JDBC-ből közvetlenül elérhető meghajtók száma viszonylag kevés.

Viszont van egy sor olyan meghajtó, amely az ismertebb adatbázisprogramok és a Microsoft által bevezetett interfész között teremt kapcsolatot.

Az ezeket tartalmazó interfész az ODBC
(***O**pen **D**ata**B**ase **C**onnectivity*)

3. PÉLDA

ODBC:

Windows-os környezetben az egyik legnépszerűbb interfész az alkalmazások és az adatbázis-kezelő rendszerek között.

Célja az, hogy lehetővé tegye bármely adat elérését bármely alkalmazásból, függetlenül attól, hogy milyen adatbázis-kezelő rendszer kezeli az adatokat.
Az ODBC egy közbülső réteget képez az adatbázis és az alkalmazás között. A réteg feladata az, hogy az alkalmazás kéréseit úgy alakítsa át, hogy az adatbázis-kezelő megértse.

Megjegyzés: A nyílt forrású adatbázis-kezelő szoftverekhez is léteznek ODBC driverek, ezek általában megtalálhatóak a telepítő csomagokban.

3. PÉLDA

JDBC-ODBC híd:

A Java-ban kifejlesztettek egy ún. JDBC-ODBC hidat, hogy a Java programozók tetszőleges adatbázishoz hozzáférhessenek.

Ez lehővé teszi, hogy valódi JDBC meghajtó helyett a számos ODBC meghajtó valamelyikét használjuk.

3. PÉLDA

Az adatbázis:

Microsoft Access - [mmegyek : tábla]

Fájl Szerkesztés Nézet Beszúrás Formátum Rekordok Es

sorrend	magyarmegyek	lakos_ezer
1	Bács-Kiskun	567
2	Baranya	434
3	Borsod-Abaúj-Zemplén	807
4	Békés	435
5	Budapest	2064
6	Csongrád	456
7	Fejér	423
8	Győr-Sopron	429
9	Hajdú-Bihar	553
10	Heves	350
11	Komárom	322
12	Nógrád	239
13	Pest	979
14	Somogy	360
15	Szabolcs-Szatmár	592
16	Jász-Nagykun-Szolnok	445
17	Tolna	268
18	Vas	205
19	Veszprém	383
20	Zala	317
(Számláló)		0

3. PÉLDA

Hozzuk létre az adatbázist:

1. Hozzuk létre a megyek.mdb adatbázist
2. Hozzunk létre az adatbázisban egy új táblázatot (mmegyek)
3. A táblázatnak legyen három oszlopa: sorrend, magyarmegyek, lakos_ezer.
4. Töltsük fel a táblázatot, és mentsük el.

3. PÉLDA

```
public class Adatbazis {

   private Connection kapcsolat;
   private Statement parancs;

   public Adatbazis() {
       kapcsolatFelepites();
       adatKiolvasas();
       kapcsolatLezaras();
   }

public static void main(String[] args) {
    new Adatbazis();
}
```

3. PÉLDA

```
private Connection kapcsolat;

private void kapcsolatFelepites() {

    String URL = "jdbc:odbc:MS Access Database;"
            + "DBQ=C:/temp/megyek.accdb";

    // Meghajto betöltese
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch (ClassNotFoundException ex) {
        System.out.println("A JDBC/ODBC meghajto nem toltheto be");
    }

    // Kapcsolat felepitese az adatbazishoz
    try {
        kapcsolat = DriverManager.getConnection(URL);
    } catch (Exception ex) {
        System.out.println("Az "+URL+" kapcsolat nem hozhato letre");
    }
}
```

Adatbázis URL

Meghajtó-program betöltése

```
private Statement allitas;

private void adatKiolvasas() {
    try {
        // Adatok kiolvasasa
        allitas = kapcsolat.createStatement();
        ResultSet adattomeg;
        adattomeg = allitas.executeQuery("SELECT*FROM mmegyek;");

        // fejlec kiiratasa
        System.out.println("\n Magyarorszag megyei \t\tlakosok(ezer)\n");

        // rekordok egyenkenti megjelenitese
        String megye;
        int lakos;

        while (adattomeg.next()) {
            megye = adattomeg.getString("magyarmegyek");
            lakos = adattomeg.getInt("lakos_ezer");
            System.out.printf("%-20s%20d\n", megye, lakos);
        }
    } catch (SQLException ex) {
        Logger.getLogger(Adatbazis.class.getName()).log(Level.SEVERE, nul
    }
}
```

3. PÉLDA

Előfordulhat, hogy így kell megadni az URL-t:

```
String URL =
      "jdbc:odbc:DRIVER=Microsoft Access Driver (*.mdb, *.accdb);"
                  + "DBQ=C:\\temp\\megyek.accdb;";
```

3. PÉLDA

```
private void kapcsolatLezaras() {
    try {
        kapcsolat.close();
    } catch (SQLException ex) {
        Logger.getLogger(Adatbazis.class.getName()).log(Level.SEVERE, null
    }
}
```

Magyarország megyéi	lakosok(ezer)
Tolna	268
Vas	205
Veszprém	383
Zala	317
Bács-Kiskun	567
Baranya	434
Borsod-Abaúj-Zemplén	807
Békés	435
Budapest	2064
Csongrád	456
Fejér	423
Győr-Sopron	429
Hajdú-Bihar	553
Heves	350
Komárom	322

3. PÉLDA

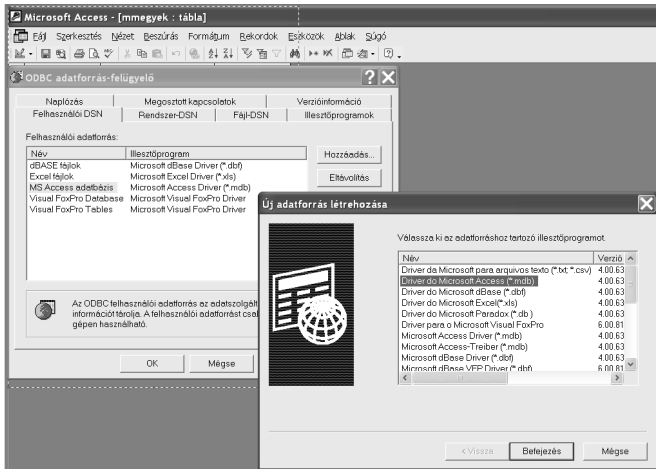
Probléma:

Csak a kezdő programozó drótozza be fixen az adatbázis helyét. ☹ (Elérési útvonalát.)

Megoldás:

Az adatbázis regisztrálása.

3. PÉLDA (Windows XP):



3. PÉLDA (Windows XP):

Állítsuk be az ODBC meghajtót:

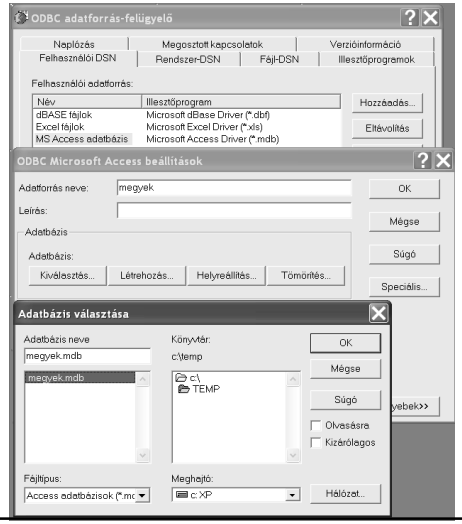
1. Nyissuk meg: Start/Vezérlőpult/Felügyeleti eszközök, és válasszuk ki az ODBC adatforrások parancsot.
2. Az ODBC adatforrás-felügyelő párbeszédablak:
   Felhasználói Dns fül → Hozzáadás →
   Új adatforrás hozzáadása → Microsoft Acces Driver
   → Befejezés

3. PÉLDA (Windows XP):

ODBC beállítása (folyt):

3. Adjunk nevet az új meghajtó-kapcsolatnak (pl. megyek)
4. Határozzuk meg, hogy a meghajtó-kapcsolat melyik adatbázishoz férhet hozzá (Kiválasztás gomb)
5. Szükség esetén a Speciális gombra kattintva felhasználónevet és jelszót is megadhatunk.
6. Zárjuk be az összes párbeszédablakot.

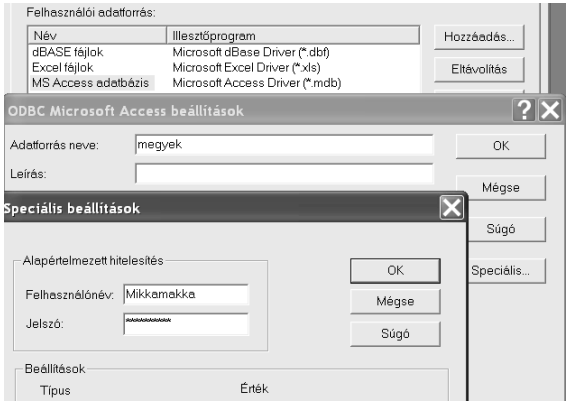
3. PÉLDA (Windows XP):



EKKOR A KAPCSOLÓDÁS

```
private Connection kapcsolat;

private void kapcsolatFelepites() {
    String URL = "jdbc:odbc:megyek";
    String felhasznalo = "Mikkamakka";
    String jelszo = "Vacskamati";

    // Meghajto betöltese
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch (ClassNotFoundException ex) {
        System.out.println("A JDBC/ODBC meghajto nem toltheto be");
    }

    // Kapcsolat felepitese az adatbazishoz

    try {
        kapcsolat = DriverManager.getConnection(URL, felhasznalo, jelszo);

    } catch (Exception ex) {
        System.out.println("Az " + URL + " kapcsolat nem hozhato letre");
    }
}
```

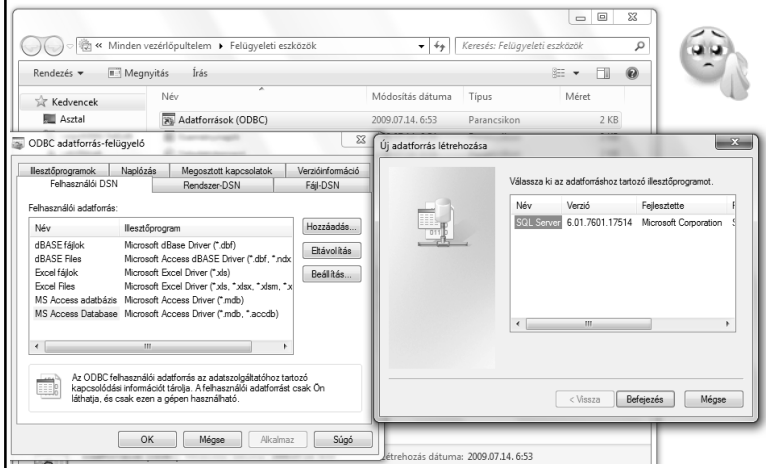
3. PÉLDA (Windows XP):



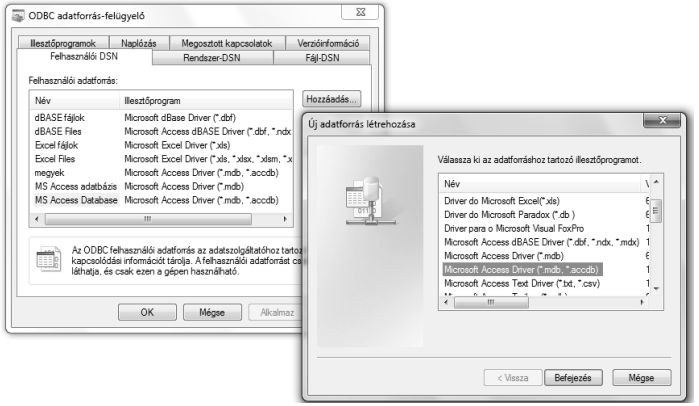
3. PÉLDA (Win 7):

Vezérlőpult – Felügyeleti eszközök – ODBC adatforrás felügyelő – MS Acces Database – Hozzáadás – …

3. PÉLDA (Win 7):



3. PÉLDA (Win 7):



3. PÉLDA (Win 7):

Némi Google

Megoldás:

„So the key is to use c:\windows\System32\odbcad32.exe”

3. PÉLDA

Feladat:

A BEMUTATOTT PÉLDA ÖNÁLLÓ KIPRÓBÁLÁSA!

(De inkább az előző kettőt próbálja. ☺)