Projekt BD

Cel przedmiotu

- Umiejętność stworzenia aplikacji korzystającej z bazy danych
  - Aplikacja graficzna
  - Pełna wersja (instalacja, testowanie)
- Środowisko: Java, Eclipse

Umiejętności nabywane

- Przygotowanie bazy danych (Data Modeler)
- Połączenie Javy z bazą danych (JDBC)
- Stworzenie warstwy DAO
- Budowa interfejsu graficznego (Swing)
- Użycie biblioteki ORM (Hibernate)
- Testowanie z użyciem JUnit
- Użycie Log4j
- Stworzenie instalatora (NSIS)

Narzędzia (darmowe)

- Oracle Data Modeler
- Java Development Kit
- Eclipse + JUnit
- Log4J
- MySQL/Derby
- Hibernate
- Nullsoft Scriptable Install System

Dostęp do baz danych z języka Java

Paweł Kasprowski

Podstawy połączenia z bazą danych

- Klient
- Serwer
- Sterownik
  - Własne API (Application Programmer Interface)

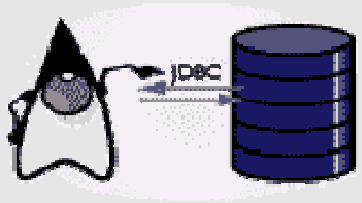
Łączenie z bazą danych

- Sterownik
- Protokół komunikacyjny
- Adres serwera
- Port nasłuchowy
- Przesłanie zapytania SQL
- Odebranie rezultatów

JDBC

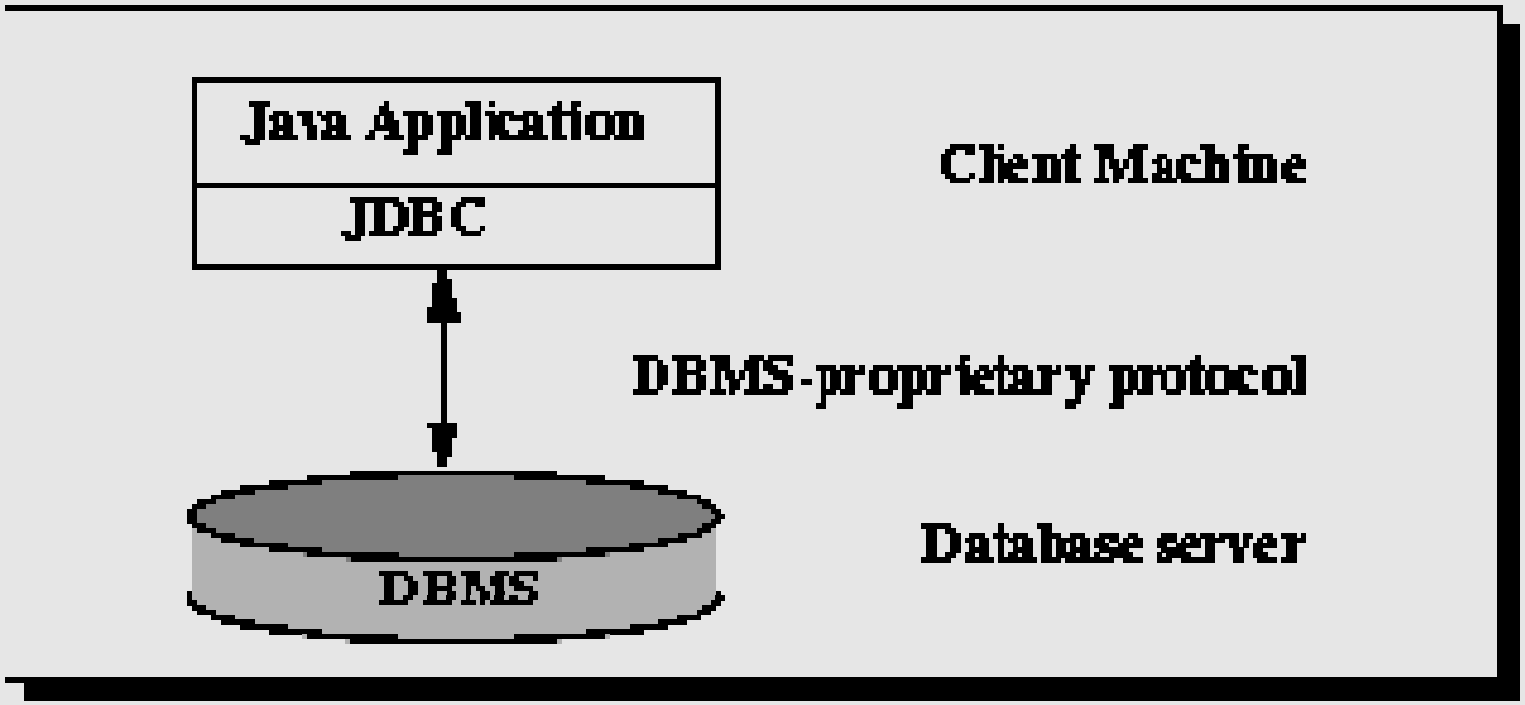
JDBC API pozwala na:

- Ustalenie połączenia z bazą
- Wysyłanie poleceń SQL
- Przetwarzanie rezultatów



Architektura JDBC

- Konieczny sterownik (plik jar)



Sposób działania JDBC

Załaduj sterownik

```
Class.forName( DriverClassName);
```

Połącz się ze źródłem danych

```
Connection con = DriverManager.getConnection(
URL, Username, Password );
```

Stwórz polecenie

```
Statement stmt = con.createStatement();
```

Zapytaj bazę danych

```
ResultSet result = stmt.executeQuery("SELECT
* FROM table1");
```

Ładowanie sterownika

Statyczne:

```
import drivername;
```

Dynamiczne:

```
Class.forName(”drivername”);
```

Na przykład:

```
com.mysql.jdbc.Driver
```

lub

```
com.microsoft.jdbc.sqlserver.SQLServerDriver
```

Połączenie ze źródłem danych

```
Connection con = DriverManager.getConnection
   ( URL, Username, Password );
```

URL = jdbc:<subprotocol>:<subname>

<subname> = //host:port/dbname;params

Na przykład:

jdbc:microsoft:sqlserver://db.psl.pl:1433

jdbc:mysql://localhost/baza

jdbc:odbc:uran

Podsumowanie

- Trzy kroki do nawiązania połączenia:
- (1) Dodanie odpowiedniego pliku jar na ścieżce aplikacji
- (2) Załadowanie klasy w aplikacji
  - Trzeba znać nazwę klasy
- (3) Nawiązanie połączenia
  - Trzeba wiedzieć jak tworzyć URL

Tworzenie polecenia (Statement)

```
Statement stmt = con.createStatement();
```

- *executeQuery()* uruchamia polecenia SQL zwracające listę wierszy jako *ResultSet*.
- *executeUpdate()* uruchamia polecenia SQL modyfikujące dane lub strukturę bazy
- *execute()* uruchamia polecenie SQL dowolnego typu (zwraca boolean gdy jest ResultSet)

ResultSet

```
ResultSet rs=stmt.executeQuery(„select * from table”);
```

- Dostęp do jednego wiersza danych jednocześnie
- *next()* – przejście do następnego wiersza
- *getXXX(i)* – wartość i-tej kolumny
- Najczęściej: *getInt(), getString(), getDate(), getDouble()*...
- Informacje o strukturze danych (nazwy i typy kolumn) - MetaData

Metody do odczytywania danych różnych typów

SQL Type	Java Method
BIGINT	`getLong()`
BINARY	`getBytes()`
BIT	`getBoolean()`
CHAR	`getString()`
DATE	`getDate()`
DECIMAL	`getBigDecimal()`
DOUBLE	`getDouble()`
FLOAT	`getDouble()`
INTEGER	`getInt()`
LONGVARBINARY	`getBytes()`
LONGVARCHAR	`getString()`
NUMERIC	`getBigDecimal()`
OTHER	`getObject()`
REAL	`getFloat()`
SMALLINT	`getShort()`
TIME	`getTime()`
TIMESTAMP	`getTimestamp()`
TINYINT	`getByte()`
VARBINARY	`getBytes()`
VARCHAR	`getString()`

Pierwszy przykład JDBC

```
import java.sql.*;
class DBExample1{
public static void main(String[] args){
  try{
    Class.forName("com.mysql.jdbc.Driver");
    Connection con =
        DriverManager.getConnection("jdbc:mysql://localhost/baza","lab","lab");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from pracownicy");
    while(rs.next())
      System.out.println(rs.getString(1)+"|"+rs.getString(4));
    con.close();
    }
  catch(SQLException ec) { System.err.println(ec.getMessage()); }
  catch(ClassNotFoundException ex) {System.err.println("Cannot find driver.");}
  }
}
```

DBExample1.java

Znajdowanie sterownika

MySQL:

```
java -classpath ".;mysql-connector-java.jar" %1
```

SQLServer:

```
java -classpath ".;mssqlserver.jar" %1
```

run.bat

Przykład JDBC – SQL Server

```
import java.sql.*;
class DBExample1{
public static void main(String[] args){
  try{
    Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
    Connection con =
        DriverManager.getConnection("jdbc:microsoft:sqlserver://localhost",
                                    "lab","lab");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from pracownicy");
    while(rs.next())
      System.out.println(rs.getString(1)+"|"+rs.getString(4));
    con.close();
    }
  catch(SQLException ec) { System.err.println(ec.getMessage()); }
  catch(ClassNotFoundException ex) {System.err.println("Cannot find driver.");}
  }
```

DBExample1.java

Przykład JDBC–ODBC Bridge

```
import java.sql.*;
class DBExample1{
public static void main(String[] args){
  try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con =
        DriverManager.getConnection("jdbc:odbc:uran","lab","lab");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select * from pracownicy");
    while(rs.next())
      System.out.println(rs.getString(1)+"|"+rs.getString(4));
    con.close();
    }
  catch(SQLException ec) { System.err.println(ec.getMessage()); }
  catch(ClassNotFoundException ex) {System.err.println("Cannot find driver.");}
  }
}
```

DBExample1.java

Zadanie

- Uruchomienie Eclipse
- Przygotowanie i uruchomienie programu

Użycie ResultSet MetaData

- Wyświetlenie nagłówków kolumn

```
ResultSet rs = stmt.executeQuery("select * from pracownicy");

int noOfCols = rs.getMetaData().getColumnCount();

for (int i=1;i<=noOfCols;i++)
  System.out.print(rs.getMetaData().getColumnName(i)+”|”);
```

Użycie ResultSet MetaData

- Wyświetlenie nagłówków kolumn

```
ResultSet rs = stmt.executeQuery("select * from pracownicy");

int noOfCols = rs.getMetaData().getColumnCount();

String colNames="";

for (int i=1;i<=noOfCols;i++)
 colNames = colNames+'|'+rs.getMetaData().getColumnName(i);

System.out.println(colNames);
```

Parametryzacja kolumn

- Wyświetlenie wszystkich zwróconych kolumn

```
while(rs.next())
  {
  for (int i=1;i<=noOfCols;i++)
    System.out.print(rs.getString(i)+”|”);
  System.out.println();
  }
```

Parametryzacja kolumn

- Wyświetlenie wszystkich zwróconych kolumn

```
while(rs.next())
  {
  String row="";
  for (int i=1;i<=noOfCols;i++)
    row = row + '|' + rs.getString(i);
  System.out.println(row);
  }
```

DBExample2.java

executeUpdate

```
int rows=stmt.executeUpdate(„update table set
   field=3 where id=5”);
```

- Dla poleceń DML i DDL
- Zwraca ilość przetworzonych wierszy
- Zwraca zero dla poleceń DDL (CREATE TABLE itp.)

PreparedStatement

```
PreparedStatement pstmt =
   con.prepareStatement("select * from table1
   where id=? and field like ?")
```

- Pre-kompilacja zapytań po stronie serwera
- Przyspiesza działanie podobnych zapytań
- Czytelniejszy kod
- Pozwala na użycie parametrów(marker ‘?’)
- setXXX() – ustawienie wartości parametru

PreparedStatement

```
   // definicja polecenia
PreparedStatement pstmt =
   con.prepareStatement("select * from table1 where
   id<? and field like ?");

   // ustawienie parametrów
pstmt.setInt(1,20);
pstmt.setString(2,"value");

   // uruchomienie
ResultSet rs = pstmt.executeQuery();
```

DatabaseMetaData

```
DatabaseMetaData dbmd = con.getMetaData();
```

- 150 metod!
- getCatalogs() – bazy danych
- getSchemas() – użytkownicy
- getTableTypes() – np. TABLE,VIEW
- getTables(catalog,schemaP,tableP,types)

Pierwsza funkcja

```
void showRS(ResultSet rs) throws SQLException
{
int noOfCols = rs.getMetaData().getColumnCount();
String colNames="";
for (int i=1;i<=noOfCols;i++)
  colNames = colNames + '|' + rs.getMetaData().getColumnName(i);
System.out.println(colNames);

while(rs.next())
  {
  String row="";
  for (int i=1;i<=noOfCols;i++)
      row = row + '|' + rs.getString(i);
  System.out.println(row);
  }
}
```

Wywołanie funkcji

```
class MetaData
{
public static void main(String[] args)
 {
 ...setting connection...
 ResultSet rs = stmt.executeQuery("select * from tb");
 showRS(rs);
 ... closing connection...
 }
...
}
```

BŁĄD! – nie można wywoływać metod obiektu z funkcji statycznej!

Static members

```
Class Foo{
  int x;
  ...
  }
...
Foo f1=new Foo();
Foo f2=new Foo();
f1.x=1;
f2.x=2;
println(f1.x+”,”+f2.x);
```

1,2

```
Class Foo{
  static int x;
  ...
  }
...
Foo f1=new Foo();
Foo f2=new Foo();
f1.x=1;
f2.x=2;
println(f1.x+”,”+f2.x);
```

2,2

Właściwe wywołanie funkcji

```
class MetaData
{
public static void main(String[] args)
 {
 MetaData md = new MetaData();
 }
public MetaData()
 {
 ...
 ResultSet rs = stmt.executeQuery("select * from tb");
 showRS(rs);
 ...
 }
...
}
```

Przykład MetaData

```
DatabaseMetaData dbmd = con.getMetaData();
ResultSet rs;
rs=dbmd.getCatalogs(); //databases
System.out.println("\n------------Bazy danych-------------");
showRS(rs);
rs=dbmd.getSchemas(); //users
System.out.println("\n------------Użytkownicy-------------");
showRS(rs);
rs=dbmd.getTableTypes();
System.out.println("\n------------Typy tabel-------------");
showRS(rs);
String[] ttp={"TABLE","VIEW"};
rs=dbmd.getTables("labbd","%","%",ttp);
System.out.println("\n--------------Tabele---------------");
showRS(rs);
```

MetaData.java

Transakcje

- Niepodzielny blok poleceń
- Utrzymanie spójności danych
- Domyślnie – AutoCommit – każde polecenie to osobna transakcja

```
con.setAutoCommit(false)
con.commit()
con.rollback()
```

Przykład transakcji (przelew bankowy)

```
...
con.setAutoCommit(false);
try{
int n1 = stmt.executeUpdate("update accounts set
   amount=amount-100 where custid=75");
int n2 = stmt.executeUpdate("update accounts set
   amount=amount+100 where custid=43");
if(n1==1 && n2==1)
  con.commit();
else
  con.rollback();
}catch(SQLException ec)
  {con.rollback();
   System.err.println(ec.getMessage());}
...
```

Zadanie

- Stworzenie aplikacji MetaData