Wykład 10

- Bazy Danych wprowadzenie
- JDBC
- komunikacja z bazą danych
- HSQLDB

Wprowadzenie — SQL

Przykładowa baza danych:

id	Imie	Nazwisko	Wiek	numer
1	Barbara	Tokarska	47	FSX458721
2	Tomasz	Czyżewski	24	HGJ874398
3	Jerzy	Pietras	35	VTF937836
4	Agata	Kałuża	17	JKD259077

Przykładowe operacje na Bazie danych:

```
SELECT Imie,Nazwisko FROM Tabela WHERE id = 3;
UPDATE Tabela SET Imie = 'Marek' WHERE id = 4;
DELETE FROM Tabela WHERE Imie = 'Tomasz';
INSERT INTO Tabela (id, Imie) VALUES (5, 'Dorota');
```

JDBC (Java Database Connectivity)

W javie z bazą danych komunikujemy się za pomocą poszczególnych klas i interfejsów.

`JDBC` — to specyfikacja określająca zbiór klas i interfejsów napisanych w Javie, które mogą być wykorzystane przez programistów tworzących oprogramowanie korzystające z baz danych

Implementacja JDBC ma być dostarczona przez producentów baz danych!

1. **Minusy**:
   Java mówi jakich klas, metod, mamy używać natomiast implementację tych metod zostawia twórcom poszczególnych baz. Jeśli twórca ją przygotował, to musimy ją samodzielnie pobrać i dołączyć do programu(zwykle jest to biblioteka *.jar*).
2. **Plusy**:
   Podczas korzystania z pobranej wcześniej implementacji zadanych metod, nie interesuje nas z jakich baz danych korzystamy(czy jest to MySQL, MsSQL, Oracle, SQLite, itp.), ponieważ korzystamy z uniwersalnych klas i metod. Nie obchodzi nas ich sposób implementacji.

> Dzięki temu ten sam program napisany w Javie może współpracować z różnymi systemami baz danych (np. Oracle, Sybase, IBM DB2). Wystarczy podmienić odpowiednie biblioteki implementujące *JDBC*.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestDriver {
    public static void main(String[] args) {
        Statement stmt = null;
        ResultSet rs = null;
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection con = DriverManager.getConnection(
                        "jdbc:mysql://localhost/test?user=monty&password=");

            stmt = con.createStatement();

            rs = stmt.executeQuery("SELECT * FROM Tabela");

            while (rs.next()) {
                System.out.println(rs.getString("Imie"));
            }
        }
        catch (Exception ex) { // obsluga bledow
        } finally { // zwalnianie zasobow
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException sqlEx) { // ignorujemy
                }
                rs = null;
            }
            if (stmt != null) {
                try {
                stmt.close();
                } catch (SQLException sqlEx) { // ignorujemy
                }
                stmt = null;
            }
            // analogicznie con.close();
        }
    }
}
```

`Class.forName("com.mysql.jdbc.Driver").newInstance();` - Ładowanie sterownika *JDBC* pobranego wcześniej od producenta — wczytanie do pamięci

Class — z pakietu java.lang

`newInstance()` — tworzy nową instancję obiektu

Po co tworzyć nową instancję obiektu i ładować go do pamięci?

Klasa `Class` musi się zarejestrować w obiekcie ***DriverManager***. Oznacza to, że jeżeli ktoś będzie chciał nawiązać połączenie z `jdbc:mysql...` to *DriverManager* przekaże to żądanie do `Class` . (Odbywa się to w kodzie statycznym)

`Connection con = DriverManager.getConnection("jdbc:mysql://localhost/test?user=monty&password=");` — ta linia nawiązuje połączenie z bazą danych *MySQL* o nazwie *"test"*, która jest uruchomiona na lokalnym serwerze (localhost). Użytkownik o nazwie *"monty"* jest używany do logowania, ale nie jest podane hasło.

To jak powinien wyglądać ten *connectString* jest opisane w dostarczonej dokumentacji!

w tym przypadku:

```
jdbc:mysql://Serwer/nazwaBazyDanych?user=NazwaUzytkownika&password=Haslo
```

`con.createStatement();` — tworzy obiekt *Statement*, który **umożliwia wykonanie zapytań** *SQL*

`stmt.executeQuery("SELECT * FROM Tabela");` - **wykonuje zapytanie** *SQL*, które wybiera wszystkie rekordy z tabeli o nazwie *"Tabela"*.

`while (rs.next()) {}` - ta pętla przechodzi przez wyniki zapytania i drukuje wartość kolumny *"Imie"* dla każdego rekordu. ***ResultSet rs* jest początkowo ustawiony przed pierwszym rekordem** (No i jak z tej budowy wynika *.next()* zwraca false gdy kolejny rekord już nie istnieje).

Nawiązywanie Połączenia

Istnieją dwie metody nawiązania połączenia z bazą danych:

1. za pomocą klasy **`DriverManager`** :

```
String url = "jdbc:odbc:bazadanych";
Connection con = DriverManager.getConnection(url, "login", "haslo");
```

2. za pomocą klasy `DataSource` i usługi *JNDI*:

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("jdbc/MojaDB");
Connection con = ds.getConnection("myLogin", "myPassword");
```

DriverManager

Jest **tradycyjną warstwą zarządzającą *JDBC*** pomiędzy użytkownikiem a sterownikiem. Aktualną listę dostępnych sterowników można uzyskać za pomocą metody:

- `Enumeration DriverManager.getDrivers()`

Aby załadować dodatkowy sterownik należy skorzystać z metody:

- `Class Class.forName(String)`

podając jako argument klasę ze sterownikiem np:

```
Class.forName("jdbc.odbc.JdbcOdbcDriver"); // protokół jdbc:odbc
Class.forName("com.mysql.jdbc.Driver"); //protokół jdbc:mysql..
```

DataSource

reprezentuje źródło danych. Zawiera informacje identyfikujące i opisujące dane. Obiekt *DataSource* współpracuje z technologią *JNDI*, jest tworzony i zarządzany niezależnie od używającej go aplikacji. Korzystanie ze źródła danych zarejestrowanego w *JNDI* zapewnia:

- brak bezpośredniego odwołania do sterownika przez aplikację
- umożliwia implementację grupowania połączeń (pooling) oraz rozproszonych transakcji.

Te cechy sprawiają, że korzystanie z klasy *DataSource* **jest zalecaną metodą** tworzenia połączenia z bazą danych, szczególnie **w przypadku dużych aplikacji rozproszonych**.

`JNDI` — (Java Naming and Directory Interface) usługa do przechowywania różnych zasobów które mogą być potrzebne w aplikacjach Javy.

Podsumowując:

- Korzysta się raczej w większych programach, jeden zbiór na wszystkie zasoby — większa wygoda
- Nie wymaga modyfikacji kodu w sytuacji zmiany Bazy Danych, wykonujemy tylko zmiany w pliku z zasobami

Przesyłanie zapytań

1. Do przesyłania zapytań do bazy danych służą obiekty klasy:
   - `Statement` — typowe pytania (bezparametrowe)
   - `PreparedStatement` — prekompilowany pytania zawierające parametry wejściowe. Używany do wykonywania większych serii zapytań gdzie różnią się one tylko jednym parametrem
     Przykład:

     ```
     SELECT * FROM Tabela WHERE id = ?;
     ```

   Po dodaniu ***?*** sterownik przygotowując takie zapytanie wie, że to zapytanie będzie używane do serii zapytań dlatego może wykonać kroki optymalizacyjne aby **wykonało się to szybciej** niż zwykłe wywołania *Statement*.
   - `CallableStatement` — służy do **wykonywania przechowywanych procedur na bazach danych**. Przechowywane procedury są funkcjami lub procedurami zdefiniowanymi w bazie danych, które mogą być wywoływane przez aplikację.

   Obiekt *Statement* tworzy się w ramach nawiązanego wcześniej połączenia:

   ```
   Connection con = DriverManager.getConnection(url, login, pass);
   Statement stmt = con.createStatement();
   stmt.executeUpdate("INSERT INTO table(name, price) VALUE 'ser', 2.0");
   ```

2. Do przesyłania zapytań służą metody:
   - `executeQuery` – pytania zwracające dane: ***SELECT***
   - `executeUpdate` – pytania zmieniające dane: ***INSERT***, ***UPDATE***, ***CREATE TABLE***, ...
   - `execute` – **dowolne zapytania**. Dzięki tej instrukcji można wykonać sekwencje pytań, przekazać i odebrać dodatkowe parametry

> W ramach jednego obiektu Statement można wykonać sekwencyjnie kilka zapytań. Po zakończeniu używania obiektu zaleca się wywołanie metody *close()*.
> Jak wykonamy sobie zakończenie połączenia to również zamykane są wszystkie *Statement*-y które były w tym połączeniu utworzone.

Odbieranie wyników

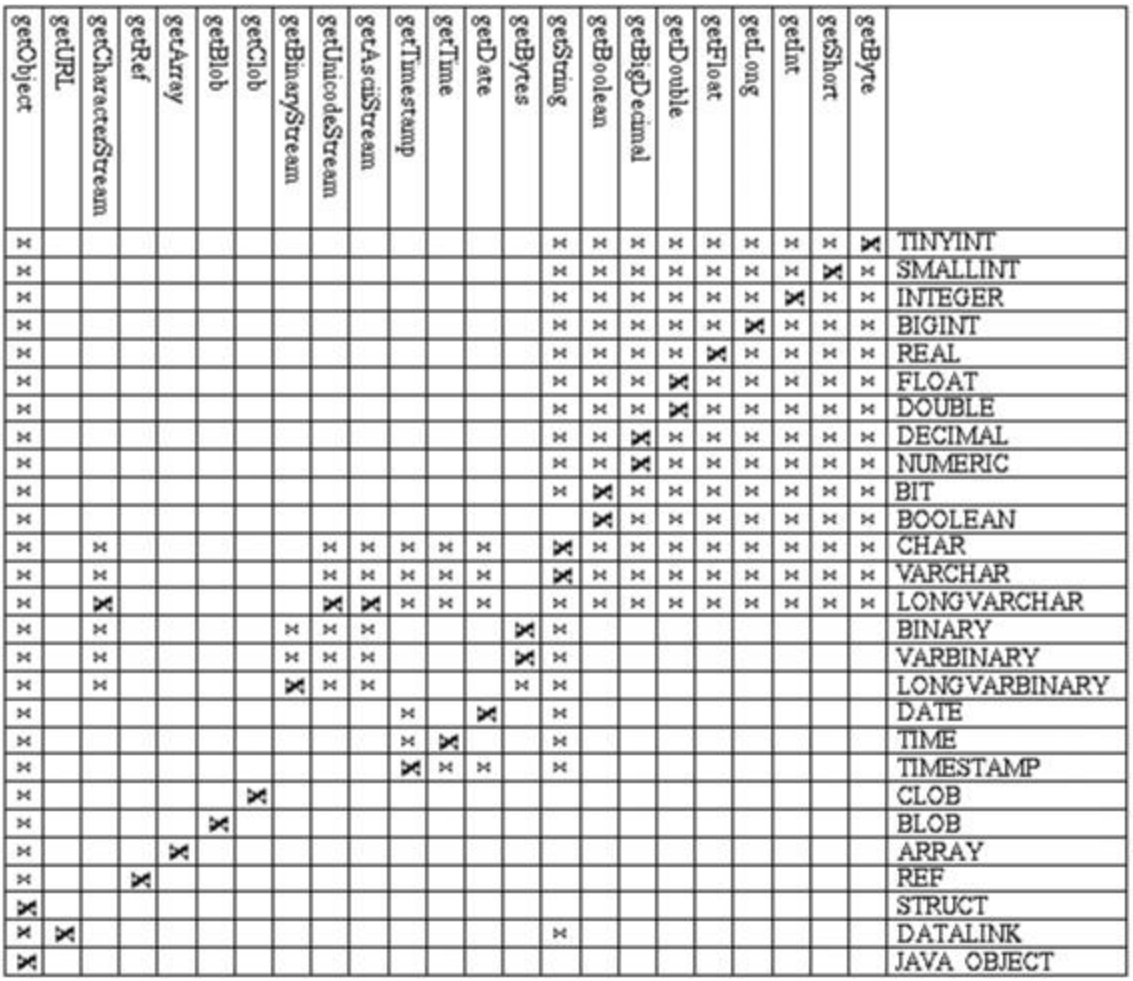
Wyniki zwrócone w wyniku wykonania zapytania są dostępne poprzez obiekt typu `ResultSet`.

Przykład:

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM table");
while (rs.next()) {
    // odebranie i wypisanie wyników w bieżącym rekordzie
    int i = rs.getInt("a");
    String s = rs.getString("b");
    float f = rs.getFloat("c");
    System.out.println("Rekord = " + i + " " + s + " " + f);
}
```

Skąd wiemy jak odbierać popszczególne pola?

- Wiemy jaka zbudowana jest tabela
- Istnieją trochę bardziej zaawansowane techniki sprawdzania jakiego typu są dane pola (Poza zakresem wykładów)
- dane typy możemy odbierać metodami do innych typów czyli np. `float` jako `int`



(Nie trzeba się jej uczyć)

Trzeba kojarzyć za to, że:

- Istnieją typy bazodanowe
- Istnieją metody w `ResultSet` 'cie do pozyskania ich, i one się tam jakoś mapują

HSQLDB (HyperSQL DataBase)

`HSQLDB` — to system zarządzania relacyjnymi bazami danych (RDBMS), napisany w języku Java. Jest to oprogramowanie *open-source*, dostępne na licencji *BSD*, co oznacza, że można je używać, modyfikować i rozprowadzać bez konieczności udostępniania kodu źródłowego projektu.

Niektóre własności:

- **obsługa *SQL*'a,**
- **obsługa transakcji** (COMMIT, ROLLBACK, SAVEPOINT), zdalnych procedur i funkcji (mogą być pisane w Javie), triggerów itp.

> Transakcje pozwalają na grupowanie operacji bazodanowych w logiczne jednostki i umożliwiają albo wykonanie ich wszystkich, albo żadnej

- **możliwość dołączania do programów i appletów**. Działanie w trybie *read-only*
- rozmiar tekstowych i binarnych danych **ograniczony przez rozmiar pamięci**
- `HyperSQL Server` – **tryb preferowany**. Klasa *org.hsqldb.server.Server*
- `HyperSQL WebServer` – używany, jeśli serwer może używać tylko protokołu *HTTP* lub *HTTPS*. W przeciwnym razie niezalecany. Klasa *org.hsqldb.server.WebServer*
- `HyperSQL Servlet` – używa tego samego protokołu co *WebServer*. Wymaga osobnego kontenera serwletów (np. Tomcat).
- `Stand-alone` – "serwer" bazy danych działa w ramach tej samej wirtualnej maszyny Javy, co korzystający z niego program "kliencki" (Nawiązując połączenie uruchamiamy serwer, a z kolei jak sie rozłączamy, serwer przestaje działać) — działa tylko w obrębie naszego programu z którego korzystamy

Połączenie — przykład:

```
try {
    Class.forName("org.hsqldb.jdbc.JDBCDriver" );
} catch (Exception e) {
    System.err.println("ERROR: failed to load HSQLDB JDBC driver.");
    e.printStackTrace();
    return;
}

Connection c =                 // tryb Server
DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", "");

Connection c =               // tryb WebServer - http
DriverManager.getConnection("jdbc:hsqldb:http://localhost/xdb", "SA", "");

Connection c =                 // tryb Stand-alone
DriverManager.getConnection("jdbc:hsqldb:file:/opt/db/testdb;shutdown=true", "SA", "");
```

Od tego co wpiszemy w *ConnectString* zależy w jakim trybie *HSQLDB* będzie działać!

`shutdown=true` — automatycznie przy rozłączeniu z bazą danych wszystkie zmodyfikowane pola powinny zostać zapisane w pliku bazy danych.

> Domyślny login dla roota to:
> *SA* — System Administrator
> domyślne hasło puste

Przykład uruchomienia bazy:

```
Connection c = DriverManager.getConnection(
            "jdbc:hsqldb:file:/opt/db/testdb", "sa", "");
```

Niewielkie bazy danych mogą być uruchamiane do pracy w pamięci operacyjnej komputera:

```
Connection c = DriverManager.getConnection(
            "jdbc:hsqldb:mem:testdb", "sa", "");
```

> W obecnej wersji *HSQLDB* istnieje możliwość jednoczesnego używania wielu "serwerów" baz danych działających w trybie *stand-alone*

Podsumowanie

Podstawowe klasy służące do interakcji z systemami bazodanowymi są udostępniane za pośrednictwem obiektów: `Connection`, `Statement` i `ResultSet`. Wszystkie te klasy należą do pakietu ***java.sql***. Klasy rozszerzające funkcjonalność *JDBC* (np. DataSource) znajdują się w pakiecie ***javax.sql***.

Korzystanie z interfejsu *JDBC* umożliwia jednolity sposób dostępu do różnych systemów bazodanowych. Jako przykład przedstawiono bazę *HSQLDB*. Jej ważną zaletą jest możliwość działania w ramach jednej Wirtualnej Maszyny Javy wraz z programem klienckim.