Práce s databází

Specifikace JDBC

Pro práci s relačními databázemi poskytuje Java rozhraní JDBC Java Database Connectivity.

Specifikace tohoto rozhraní má několik verzí. Verze 1.x je součástí SDK již od verze 1.1. Všechny třídy a rozhraní této specifikace jsou součástí balíčku **java.sql**. Další verze 2.x a nejnovější 3.x obsahují další třídy a rozhraní umístěné v balíčku **javax.sql.** Tento balíček je součástí SDK od verze 1.4. Pro specifikaci 2.x i 3.x byly upraveny a rozšířeny i třídy a rozhraní v balíčku java.sql.

Ovladače JDBC

Jako ovladač JDBC je označována implementace rozhraní Driver a dalších rozhraní z balíčku java.sql a dalších pomocných tříd, které zajišťují přístup k databázi. Ovladač je obvykle zabalen do jar archivu. Ovladače je možné získat jak od výrobce databáze tak od dalších dodavatelů.

Rozlišují se čtyři základní typy ovladačů JDBC:

Typ 1- most JDBC-ODBC

První možnost, jak se připojit k databázi, je využití zdroje dat ODBC. Ovladač pro tento most je jako jediný součástí SDK. Nevýhodou tohoto spojení je, že na klientské stanici musí být nainstalován příslušný ODBC driver. Navíc výkon tohoto propojení je poměrně nízký, protože každá operace musí být vykonána oběma ovladači.

Typ 2 - připojení prostřednictvím kódu nativního klienta pro přístup k síti

Ovladač tohoto typu je tvořen kódem napsaným v Javě a nativním kódem a komunikuje s klientským softwarem používaného databázového systému. Je tedy nutné aby na klientské stanici byl nainstalován příslušný software.

Typ 3 připojení prostřednictvím vrstvy aplikačního serveru

Tento typ ovladače je napsán kompletně v Javě. Posílá požadavky na server serverové komponentě ovladače, která je odpovědná za převod do formátu dané databáze. Na serveru tedy musí tato komponenta existovat. Výhodou tohoto řešení je to, že může dojít ke změně databáze bez jakékoli změny klienta, pouze na serveru je zprovozněna jiná serverová komponenta pro připojení.

Typ 4 přímé spojení s databází

Ovladač typu 4 je celý napsán v Javě a komunikuje přímo s příslušnou databází. Tento typ ovladače se používá velmi snadno, může být přímo součástí balíčku aplikace. Na serveru ani na klientovi nevyžaduje existenci žádného dalšího software.

U každého ovladače je potřeba v dokumentaci zjistit, jakou verzi specifikace JDBC podporuje. Možnosti použití jsou též ovlivněny vlastními možnostmi databáze, ke které se připojujete.

Jak ovladač připojit k aplikaci.

Zaregistrovat ovladač u aplikace můžeme pomocí metody **forName(String className)** třídy **Class** a metody **newInstance()** nebo přímo registrací instance driveru u objektu **java.sql.DriverManager** statickou metodou **registerDriver.** Jméno třídy ovladače zjistíme z dokumentace příslušného driveru. Pokud není třída uvedeného jména nalezena, je vyhozena výjimka **ClassNotFoundException** a při volání metody newInstance() výjimka **InstantiationException**, které je nutné ošetřit. Nezapomeňte, že při použití jakéhokoli balíčku mimo standard je třeba uvést umístění tohoto archivu do **CLASSPATH** nebo do parametru classpath při spouštění aplikace.

Jak navázat spojení s databází

Připojení k databázi si v aplikaci vyžádáte pomocí statické metody **getConection()** třídy **DriverManager**. Tato metoda vrací instanci **Connection**. Jako vstupní parametry této metody uvádíme nejčastěji URL databáze, uživatelské jméno a heslo pro přístup do databáze. V případě, že se nepodaří spojení navázat, je vyhozena výjimka **SQLException**.

Jak získat informace o databázi

Když je vytvořeno spojení do databáze, je možné zavolat static metodu getMetaData(), která vrací odkaz na instanci rozhraní DatabaseMetaData. V tomto rozhraní je definována celá řada metod, pomocí kterých lze získat informace o databázi (maximální počet současně otevřených připojení, zda databáze podporuje transakce, jaké typy ResultSetu jsou podporovány apod.) a o obsahu databáze (seznam tabulek, údaje o indexech atd.).

Jak zadávat jednotlivé příkazy DML, DDL a SQL

Po připojení k databázi je možno vytvořit instanci rozhraní **Statement**, tato instance pak umožňuje zadávání SQL dotazů. Instanci rozhraní Statement získáte voláním metody **createStatement(),** je to metoda z rozhraní Connection.

Jednu instanci rozhraní Statement lze opakovaně použít pro spuštění mnoha dotazů SQL. V jednom okamžiku může instance rozhraní Statement vykonávat pouze jeden dotaz. Pokud tedy v aplikace požadujete souběžné provádění několika dotazů, je třeba vytvořit odpovídající počet instancí rozhraní Statement.

Metoda createStatement() je přetížená. Pomocí parametrů, které jsou konstantami rozhraní ResultSet, lze nastavit, jaký typ ResultSetu mají vracet volání metody executeQuery().

Rozhraní Statement poskytuje několik metod pro zadávání dotazů SQL.

metoda executeUpdate()

Jako parametr této metody uvedete String obsahující DML příkaz (např. INSERT, DELETE nebo UPDATE). Metoda vrací v typu int počet řádků, které byly příkazem ovlivněny. Pokud použijete některý z příkazů jazyka DDL (např. CREATE TABLE) vrací metoda vždy 0. V případě chyby vyhodí výjimku **SQLException**.

metoda executeQuery()

Pro vykonání dotazu je třeba použít metodu executeQuery(), která má jako parametr String s SQL dotazem. Metoda vrací objekt typu **ResultSet**, který si podrobně rozebereme později. V případě neúspěšného volání dotazu vyhodí **SQLException**.

metoda execute()

Ve speciálních případech, kdy odpovědí na jeden SQL dotaz není jeden ResultSet nebo jeden údaj o počtu ovlivněných řádků, použijeme metodu execute(). Jako parametr opět uvedeme String s SQL dotazem. Metoda vrací hodnotu typu boolean, která říká jestli prvním v řadě očekávaných výsledků je ResultSet (pak vrátí true) nebo int (pak vrátí false). Návratovými hodnotami lze procházet pomocí metody **getMoreResult()** a hodnoty získávat pomoci metod **getResutSet()** a **getUpdateCount()**.

metoda addBatch()

Je metodou, která odpovídá specifikaci JDBC 2.x. Pomocí této metody je možno spojit do jedné dávky několik SQL příkazů, které by jinak byly prováděny postupným voláním metody executeUpdate(). Parametrem je tedy String s SQL příkazem.

metoda executeBatch()

Tato metoda slouží k provedení dávky, do které jsme jednotlivé příkazy přidali pomocí metody addBatch(). Metoda vrací pole hodnot int s výsledky jednotlivých příkazů (počty ovlivněných řádek). Jestliže u některého příkazu dojde k chybě, je vyhozena výjimka **BatchUpdateException** a ovladač může nebo nemusí pokračovat v provádění dalších dotazů.

Kromě rozhraní Statement jsou definováni ještě dva potomci tohoto rozhraní. PreparedStatement a CallabledStatement. Pokud máme v aplikaci velké množství velmi podobných dotazů, které se od sebe liší pouze hodnotami některých parametrů, je vhodné použít potomka rozhraní Statement **PreparedStatement**. Instanci PreparedStatement vytvoříte pomocí metody **prepareStatement()**, jako parametr zde uvedeme parametrizovaný dotaz do databáze. Stejně jako metoda createStatement() je tato metoda přetížená a umožňuje nastavit typ výsledkové sady, kterou pak provedení dotazu vrátí. Pomocí metod **setXxx()**, kde Xxx je typ jakého je dosazovaná hodnota, se pak nahradí parametry skutečnou hodnotou. Pak se volá metoda executeQuery() nebo executeUpdate() pro zaslání dotazu do databáze.

Rozhraní **CallableStatement** umožňuje spouštění uložených procedur v databázi. Lze jej vytvořit pomocí metody **prepareCall()** definované na rozhraní Connection.

ResultSet

Typy ResultSetu

Instance typu ResultSet jsou výsledkem spuštění metody executeQuery(), případně metody execute(). Je to vlastně zapouzdření dat vrácených dotazem. ResultSet může být buď jednosměrný nebo obousměrný. Lze také určit citlivost na změny v databázi, tj. zda se do vytvořeného ResultSetu promítají změny, které provedli v databázi jiní uživatelé. Výsledný ResultSet může také být jen pro čtení a nebo aktualizovatelný. Ovladače odpovídající specifikaci JDBC 1.x umějí vracet pouze jednosměrné sady výsledků s přístupem pouze pro čtení. Ovladače odpovídající specifikaci JDBC 2.x někdy nepodporují tvorbu obousměrných sad. Jaký ResultSet chceme získat jako výsledek dotazu, je třeba určit při vytváření instance typu Statement pomocí parametrů metody createStatement. Volání této metody bez parametrů by mělo vracet jednosměrný a neaktualizovatelný ResultSet, ale např. ovladač pro MySQL vrací i v tomto případě obousměrnou výsledkovou sadu. Jaké jsou možnosti nastavení výsledkových sad je uvedeno v následujících tabulkách.

Konstanty pro nastavení možnosti procházení výsledkové sady a citlivosti na změny.

Konstanta	popis
ResultSet.TYPE_FORWARD_ONLY	Vytvářeny jsou pouze jednosměrné výsledkové sady, ve kterých lze pro pohyb použít pouze metodu next()
ResultSet.TYPE_SCROLL_INSENSITIVE	Vytvářená výsledková sada je obousměrná, ale neodráží změny, které mezitím vznikly v databázi.
ResultSet.TYPE_SCROLL_SENSITIVE	Vytvářeny jsou obousměrné, na změny citlivé výsledkové sady.

tabulka 1 Konstanty třídy ResultSet pro nastavení možností procházení a citlivosti na změny

Konstanty pro nastavení možnosti úpravy dat prostřednictvím výsledkové sady.

Konstanta	popis
ResultSet.CONCUR_READ_ONLY	Výsledková sada je určena pouze pro čtení
ResultSet.CONCUR_UPDATABLE	Výsledková sada umožní i úpravu dat a její zpětné promítnutí do databáze.

tabulka 2 Konstanty třídy ResultSet pro nastavení možností zpětné aktualizace

Pohyb po ResultSetu

Pro procházení jednosměrné sady je poskytována metoda **next()**, která vrací hodnotu typu boolean. Pokud je proveden posun za poslední řádek, je vrácena hodnota false. V obousměrné sadě je kromě metody next() možno použít i metody **absolute (int row)**, **first()**, **last()**, **relative (int row)**, **previous(), beforeFirst()** a další.

Čtení hodnot z aktuálního záznamu.

Pro získání hodnot jednotlivých položek aktuálního záznamu poskytuje ResultSet metody **getXxx()**. např. **getString()**, **getObject()**. Pro každý typ dat existují dvě varianty metody. První s jedním s parametrů typu int a druhou s parametrem typu String. Varianta s int slouží pro číselné označení sloupce, jehož údaj chceme přečíst. Pozor, index prvního sloupce není 0, ale 1. Druhá varianta nám umožňuje odkaz na sloupec pomocí jeho jména.

Úprava dat v ResultSetu

Jestliže ovladač umožňuje aktualizaci dat přímo přes resultset, máme k dispozici celou řadu metod **updateXxx()** pro různé datové typy. Metody updateXxx() jsou definovány ve dvou variantách pro zadání sloupce a to buď indexem nebo názvem. Dále je možné přidat řádek metodou **insertRow()** a nebo zrušit řádek metodou **deleteRow()**. Jsou k dispozici i metody **rowDeleted()**, **rowInserted()** a **rowUpdated()**, které vracejí true nebo false. Promítnutí změn do databáze proběhne po spuštění metody **updateRow().**

Rozhraní ResultSetMetaData

Referenci tohoto typu získáme pomocí metody **getMetaData()** rozhraní ResultSet. Toto rozhraní nám pak poskytne další informace o příslušné datové sadě. Můžeme zjistit počet sloupců (metoda **getColumnCount()**) nebo názvy jednotlivých sloupců (metoda **getColumnName()**).

Podpora transakcí

Pokud databáze i ovladač podporují transakce, můžeme pro jejich zpracování používat metody **setAutoCommit()**, **commit()** a **rollback()** z rozhraní **Connection**. Jakou úroveň isolace transakcí požadujeme, nastavíme pomocí metody **setTransactionIsolation()**. Lze nastavit jen takovou isolaci, kterou používaná databáze podporuje. Bližší informace naleznete v dokumentaci JDBC, driveru a databáze.

Výjimky SQLException

Třída SQLException nám poskytuje několik metod pro podrobnější zjištění o jakou chybu se jedná. Kromě standardních **getMesage()** a **printStackTrace()** můžeme použít i metody **getErrorCode()** nebo **getSQLState()**.

Metoda getErrorCode() vrací číselný kód chyby. Přesný význam zjistíme s dokumentace k driveru nebo databázi. Metoda getSQLState vrací String obsahující kód chyby podle standardu X/OPEN SQL.

Uvolňování zdrojů

Při ukončení práce s databází se nespoléhejte na Garbage Collection, tj. neukončujte spojení dosazením hodnoty null do proměnné typu Connection. K ukončení spojení slouží metoda **close().**

Příklad připojení pro databázi MySQL

Budeme se připojovat pomocí ovladače typu 4 odpovídajícího specifikaci 3.0. Tento ovladač je volně ke stažení na stránkách www.mysql.com . Ačkoli se jedná o ovladač napsaný podle specifikace 3.0, neumožňuje provádět přímé změny v databázi prostřednictvím instance ResultSet v případě, že v dotazu byl použit join z více tabulek. Samotný ovladač je jar archiv.

Příkaz Class.forName() je třeba volat následujícím způsobem:

```
    Class.forName("com.mysql.jdbc.Driver").newInstance();
```

Nezapomeňte na to, že ovladač je jar archiv, a vy musíte mít nastavenou CLASSPATH.

Pro vytvoření připojení je třeba, jako parametr metody getConnection(), zadat řetězec s následujícím obsahem (připojujeme se k databázi telefony na serveru kitlab.vse.cz, uživatel se jmenuje java a heslo je „heslo“):

```
"jdbc:mysql://kitlab.vse.cz/telefony?user=uživatelskéJméno&password=
heslo"
```

Následující část třídy Pripojeni slouží pro připojení k databázi Telefony. Tato třída implementuje návrhový vzor Singleton, aby bylo pro aplikaci zaručeno, že existuje pouze jediné připojení k databázi. V následující podkapitole bude popsáno propojení ResutSetu s komponentou JTable grafického uživatelského rozhraní.

```
import java.sql.*;
import java.util.*;

public class Pripojeni {
/**
 * Proměnná instance bude obsahovat jedinou instanci třídy Pripojeni
*/
private static Pripojeni instance = null;

private Connection con;
private Statement statement;

/**
*   Metoda getInstance() vrací jedinou instanci třídy Pripojeni.
*/
public static  Pripojeni getInstance() {
    if (instance == null) instance = new Pripojeni();
    return instance;
}
/**
*  Private konstruktor zajišťuje, že není možné vytvořit
*  instanci jinak, než při prvním volání
*  metody getInstance. Vytváříme připojení a statement.
*/

private Pripojeni() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
    }
    catch (ClassNotFoundException e) {
        System.out.println ("Ovladac nenalezen");
        System.exit(1);
    }
    try {
         con = DriverManager.getConnection(
        "jdbc:mysql://kitomt.vse.cz/telefony?user=java
        &password=heslo");
        statement = con.createStatement();
    }
    catch (SQLException e ) {
        System.out.println ("Nepovedlo se pripojeni");
        System.out.println (e.getSQLState());
        System.exit(2);
    }
}
```
……

Příklad připojení pro databázi Oracle9

Pro připojení k databázi Oracle lze použít ovladač, který je volně ke stažení na www.oracle.com. Jedná se o ovladač typu 4 podle specifikace 3.0.

Ovladač lze registrovat metodou registerDriver() třídy DriverManager s parametrem new oracle.jdbc.OracleDriver()

```
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
```

nebo Class.forName() s parametrem oracle.jdbc.OracleDriver.

```
Class.forName("oracle.jdbc.OracleDriver");
```

Připojení ke školní instalaci Oracle na serveru kitmaster.vse.cz se SID databáze ora9, kde je nastaven zkušební uživatel student s heslem sql, je možné následujícím příkazem:

```
con=DriverManager.getConnection("jdbc:oracle:thin:@kitmaster.vse.cz:
1521:ora9","student", "sql");
```

Další práce s instancí Connection je pak identická jako při práci s MySQL.

Zobrazení dat z databáze pomocí třídy JTable.

Třída JTable je komponenta grafického uživatelského rozhraní z knihovny Swing, která zobrazuje tabulku. Ve starší knihovně AWT nenajdete její jednodušší protějšek.

Třída JTable je ze základního swingového balíčku **javax.swing**, ale pro její použití je třeba importovat ještě dva další balíčky. Je to balíček **javax.swing.table**, ve kterém najdete model pro popis obsahu jednotlivých buněk, a balíček **javax.swing.event**, ve kterém jsou třídy a rozhraní pro sledování událostí.

Pro vytvoření tabulky musíte vytvořit třídu, která popisuje, co má být obsahem jednotlivých buněk tzv. model tabulky. Tato třída musí být buď implementací rozhraní TableModel nebo být potomkem třídy AbstractTableModel.

Pokud chceme tabulku použít pro zobrazení údajů získaných dotazem do databáze, použijeme jako zdroj dat obousměrný resultset. Jestliže ovladač, který používáte, neumožňuje vytvoření obousměrného resultsetu, je nutné nejprve jeho obsah vložit do jiné struktury, nejlépe do seznamu typu ArrayList.

Nejprve je třeba popsat model pro tabulku. Bývá obvyklé vytvářet třídu s modelem pro konkrétní tabulku jako vnitřní třídu třídy, které představuje okno aplikace. Pouze v případě, že chceme napsat obecný model pro tabulku na základě jakéhokoli obousměrného resultsetu, má smysl vytvořit model tabulky jako samostatnou třídu..

Model jednoduché tabulky s needitovatelnými buňkami vytvoříme nejlépe tak, že jej bude představovat třída, která bude potomkem abstraktní třídy AbstractTableModel.

Musíte implementovat následující tři metody:

- public int getRowCount();
- public int getColumnCount();
- public Object getValueAt(int row, int column);

Metoda getRowCount() udává aktuální počet řádků v tabulce. Při navázání tabulky na resultset tedy musíme zjistit počet řádků v resultsetu. Rozhraní ResultSet neposkytuje žádnou metodu, která by to uměla přímo. Je nutné použít metodu last() a poté metodu getRow(), která vrátí číslo aktuálního řádku.

Metoda **getColumnCount()** udává počet sloupců. Můžeme uvést konstantu, pokud tvoříme model pro konkrétní dotaz, nebo pomocí metody getMetaData() získat instanci rozhraní ResultSetMetaData, a pak použít metodu getColumnCount().

Metoda getValueAt() pro jednotlivé buňky určuje obsah. Z resultsetu je získáme pomocí metody absolute() (nastavení se na řádek) a pak metodou getXxx(), nejčastěji getString(), získáme data pro jednotlivé sloupce. Pozor ale na rozdílnost v indexování JTable a ResultSetu. První buňka JTable má index 0,0 zatímco v resultsetu 1,1.

Mimo tyto povinné metody doporučuji ještě implementovat metodu public String getColumnName(), která nastaví názvy sloupců. Pro její naplnění použijte buď pole Stringů nebo při vytváření obecného popisu tabulky metodu getColumnName() ze třídy ResultSetMetaData. Následující kód ukazuje, jak napsat třídu modelu tabulky pro jakýkoli resultset.

```
class ModelTabulkyCiselnik extends AbstractTableModel {
    private ResultSet rs;

    ModelTabulkyCiselnik(ResultSet rs){
        this.rs = rs;
    }

    public int getColumnCount() {
        try{
            return rs.getMetaData().getColumnCount();
        }
        catch (SQLException e){
            System.out.println(e);
            return 0;
        }
    }

    public int getRowCount() {
        try {
            rs.last();
            return rs.getRow();
        }
        catch (SQLException e) {
            System.out.println(e);
            return 0;
        }
    }

    public String getColumnName(int column) {
        try{
            return rs.getMetaData().getColumnName(column+1);
        }
        catch (SQLException e){
            System.out.println(e);
            return null;
        }
    }
    public Object getValueAt(int row, int column) {
        try {
            rs.absolute(row+1);
            return rs.getString(column+1);
        }
        catch (SQLException e) {
            System.out.println(e);
            return null;
        }
    }
}
```