Kurs programowania aplikacji bazodanowych

Wykład 1

Paweł Rajba

Instytut Informatyki
Uniwersytet Wrocławski

Plan wykładu

- Architektura aplikacji
  - Jednowarstwowa
  - Dwuwarstowa
  - Rozproszona
- Java Database Connectivity (JDBC)
  - Wprowadzenie
  - Architektura JDBC
  - Sterowniki, sterowniki dla Oracle
  - Określanie źródła danych
  - Hierarchia klas JDBC, wybrane klasy i interfejsy JDBC
  - Uruchomienie aplikacji
  - Transakcje
  - Operowanie na wyniku
  - Preparowany SQL
  - Wykonywanie procedur składowanych
  - Wsady, Metadane

Architektura jednowarstwowe

- Charakteryzują się składowaniem bazy danych lokalnie
- Są stosunkowo łatwe do napisania i małe
- Nie są skalowalne i nie obsługuję wielodostępu
- Przykładowe rozwiązania: MS Access, Paradox

Architektura dwuwarstowe

- Są też nazywane aplikacjami typu klient–serwer
- Jest to bardzo często wykorzystywany model
- W tym modelu klienci mogą pracować równolegle
- Pojawia się problem „rozbudowanych klientów”
  - Rozwój programu staje się z czasem coraz trudniejszy
- Są stosunkowo łatwe do napisania i wzdrożenia
- Lista punktów, z których spełnienie chociaż jednego sprawia, że warto rozpatrzyć rozwiązanie typu klient–serwer
  - Czas wdrożenia jest ważniejszy od architektury aplikacji
  - Aplikacja korzysta z jednej bazy danych
  - System bazy danych jest na jednym serwerze
  - Rozmiar bazy danych będzie podczas używania aplikacji stały
  - Liczba użytkowników będzie podczas używania aplikacji stała

Architektura rozproszona

- Pozwala podzielić aplikację na wartwy
- Umożliwia „odchudzenie” klienta, ułatwia skalowanie aplikacji
- Przykładowy zestaw warstw
  - Warstwa interfejsu użytkownika
    - np. aplikacja SWING lub przeglądarka WWW
  - Warstwa generowania zawartości
    - np. tworzenie strony WWW do przekazania użytkownikowi
  - Warstwa zarządzania treścią
    - np. CMS
  - Warstwa usług WWW, WebServices
    - za zadanie będzie miała dostarczanie danych innym aplikacjom
  - Warstwa logiki biznesowej
  - Warstwa uwierzytelnienia
  - Warstwa pamięci masowej, czyli baza danych

Wprowadzenie

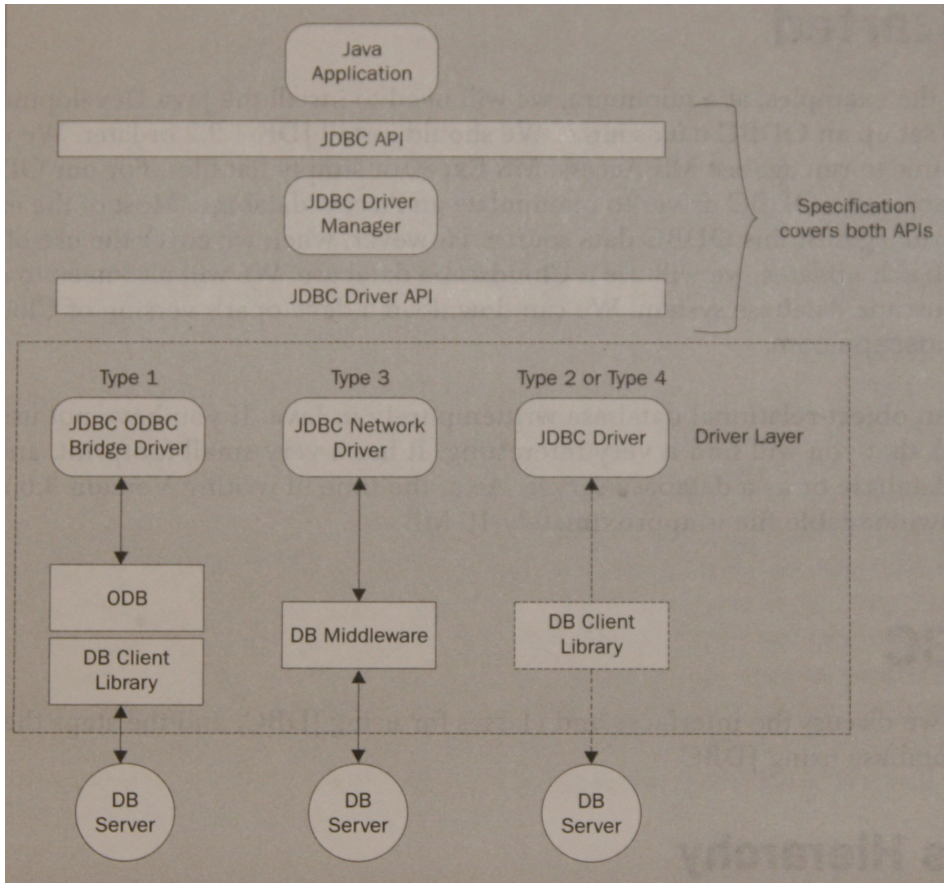
Co to jest JDBC?

- Interfejs Javy do komunikacji z DBMS
- Podstawą był ODBC, którego dostosowanie do Javy zostało zarzucone
  - ODBC było napisane w C
  - ODBC źle się integrowało z Javą
  - ODBC wymagało uruchomienie zewnętrznego natywnego kodu

Zalety JDBC

- Łatwe w użyciu
- Dobrze zintegrowane ze środowiskiem Javy
- Niezależne od bazy danych
- Niezależne od platformy systemowej

Architektura JDBC



Źródło: Dany Ayers, et al., Professional Java Data, Wrox 2001

Sterowniki JDBC

Krótki wstęp

- Niektóre sterowniki są dostępne w samej platformie Javy
- Większość sterowników jest jednak dostarczanych przez producenta DBMS
- Niestety czasami sterowniki do komercyjnych DBMS są płatne

Pod adresem http://developers.sun.com/product/jdbc/drivers można

- sprawdzić, jest sterownik określonego typu do danego DBMS
- wypełnić formularz informujący o nowym sterowniku (nowym typu), którego nie ma na liście
- dokonać edycji informacji o sterowniku (dla producentów)

Sterowniki JDBC

Mamy 4 rodzaje sterowników JDBC:

- Typ 1: JDBC-ODBC bridge driver
  - Wywołania funkcji JDBC są tłumaczone na wywołania funkcji ODBC
  - Sterownik dołączony w instalacji Java 2 SDK
  - Wymagają instalacji u klienta dodatkowego oprogramowania (ODBC)
  - Wydajność oparta na takim poziomie jest na niskim poziomie
  - W praktyce rzadko używane rozwiązanie
- Typ 2: Java plus native code driver
  - Ten sterownik jest wydajniejszy od sterownika typu 1
  - Sterownik napisany w Javie, jednak do komunikacji z DBMS jest wykorzystywany dodatkowy sterownik napisany w np. C
  - Niestety również wymagają instalacji dodatkowego oprogramowania po stronie klienta

Sterowniki JDBC

Mamy 4 rodzaje sterowników JDBC (cd.):

- Typ 3: JDBC-Net pure Java driver:
  - Wywołania JDBC są tłumaczone na niezależny od DBMS protokół, które są przetwarzane komponent pośredni
  - Sterownik łączy się bezpośrednio z tym serwerem pośredniczącym
  - Serwer pośredniczący kontynuuje komunikację z DBMS
  - Nie wymaga instalacji dodatkowego oprogramowania po stronie klienta
- Typ 4: Proprietary protocol pure Java Driver:
  - Sterownik łączy się bezpośrednio z DBMS
  - Nie ma potrzeby instalacji dodatkowego oprogramowania po stronie klienta
  - Najczęściej dostępne u producenta danego DBMS
  - Sterowniki nazywane także sterownikami typu *thin*.

Sterowniki JDBC dla Oracle

Są dwa rodzaje sterowników

- THIN
  - jest prostszy w instalacji — wystarczy skopiować plik .jar
  - przy połączeniu podajemy pełną ścieżkę do bazy danych
  - sterownik typu 4
- OCI
  - zwykle wymaga instalacji klienta — łączymy się przez podanie identyfikatora określonego w TNSNAMES
  - sterownik typu 2

My będziemy korzystać ze sterownika THIN, który znajduje się w:

`C:\OracleXE\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar`

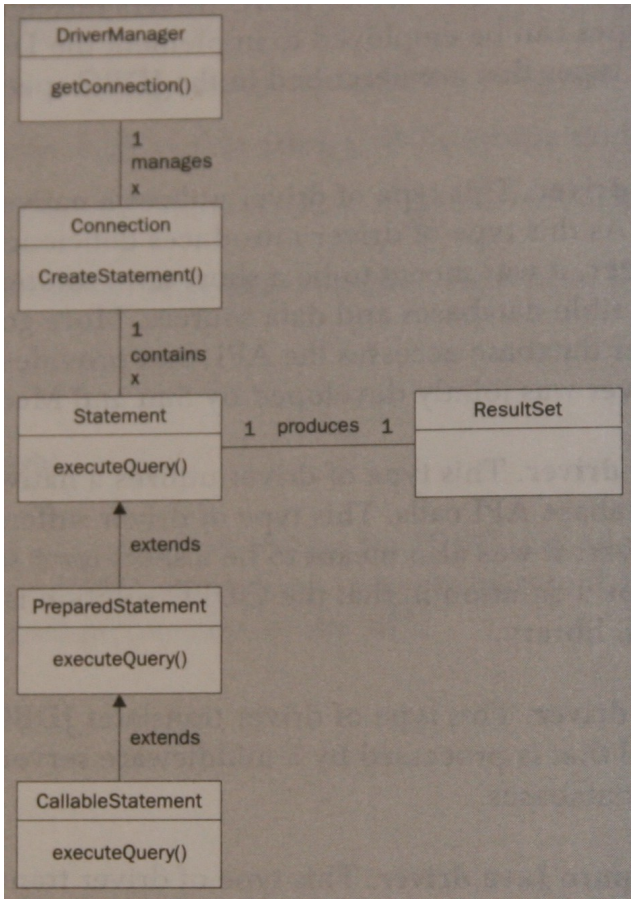
Sterowniki JDBC dla SQL Server

- Sterownik można pobrać ze strony pod adresem:
  http://www.microsoft.com/downloads/details.aspx?FamilyId=F914793A-6FB4-475F-9537-B8FCB776BEFD&displaylang=en
- Sterownik pod powyższym adresem jest typu 4
- Do sterownika dołączona jest dokumentacja i przykłady

Określanie źródła danych

- Źródło danych jest określane za pomocą składni typu URL
- Podstawowa składnia URLa jest następująca
  `jdbc:<podprotokół>:<lokalizacja-bd>`
- Znaczenie poszczególnych części:
  - subprotokół – określa rodzaj sterownika lub ogólniej sposób łączenia się z bazą danych
  - lokalizacja-bd – określa lokalizację bazy danych (format czasami zależy od subprotokołu)
- Przykładowe URLe
  - `jdbc:odbc:bazooka`
  - `jdbc:msql://serwer.bazydanych.pl:1234/baza`
  - `jdbc:mysql://localhost/test`
  - `jdbc:oracle://cos:1521/baza`
  - `jdbc:sqlserver://localhost:1433;databaseName=baza;user=sa;password=haslo`

Hierarchia klas JDBC



Źródło: Dany Ayers, et al., Professional Java Data, Wrox 2001

Wybrane klasy i interfejsy JDBC

- `java.sql.Driver`
  - interfejs odpowiedzialny za nawiązanie połączenia oraz komunikację z DBMS
  - przezroczysty dla programisty (chyba, że chcemy napisać własny sterownik do DBMS)
- `java.sql.DriverManager`
  - zarządza listą dostępnych sterowników do DBMS
  - udostępnia aplikacji niezbędnych komponentów do połączenia się z DBMS
  - posiada metody registerDriver() i deregisterDriver(), które pozwalają na rejestrację i derejestrację implementacji interfejsu `java.sql.Driver`

Wybrane klasy i interfejsy JDBC

- `java.sql.Connection`
  - służy do nawiązania połączenia z DBMS
  - reprezentuje pojedynczą transakcję
- `java.sql.Statement`
  - reprezentuje zapytanie SQL
  - ma trzy metody do wykonania zapytania
    - ResultSet executeQuery(String)
    - int executeUpdate(String)
    - int execute(String)
- `java.sql.PreparedStatement`
  - reprezentuje zapytanie wykonywane wielokrotnie
- `java.sql.CallableStatement`
  - reprezentuje wywołanie procedury składowanej

Wybrane klasy i interfejsy JDBC

- `java.sql.ResultSet`
  - reprezentuje wynik zapytania SQL
  - udostępnia metody do pobrania danych z wyniku
    - metody są postaci X getX(parametr), gdzie parametrem jest nr kolumny lub nazwa kolumny np. int getInt(int)
  - czasami jest możliwa automatyczna konwersja, np. datę można pobrać metodą getString()
  - udostępnia metody do nawigacji po wierszach wyniku np. next(), previous(), itd.
- `java.sql.Types`
  - udostępnia zestaw stałych odpowiadających typom języka SQL
  - stałe te są typu integer
  - skojarzenia bazują na specyfikacji XOPEN SQL

Wybrane klasy i interfejsy JDBC

- `java.sql.SQLException`
  - reprezentuje wyjątki powstałe w wyniku współpracy z DBMS
  - informację są w zgodzie ze specyfikacją XOPEN SQL
- `java.sql.SQLWarning`
  - reprezentuje ostrzeżenie (błąd, który nie powoduje przerwania działań)
  - poprzez metodę ResultSet.getWarnings() mamy dostęp do listy ostrzeżeń

Uruchomienie aplikacji

W środowisku Oracle JDeveloper

- Tworzymy aplikację, projekt
- Zaznaczamy projekt, wybieramy „Project Properties”
- Wybieramy w drzewie pozycję „Libraries”
- Następnie przycisk „Add Jar/Directory” i wskazujemy na sterownik do Oracle’a

W środowisku Eclipse

- Z menu kontekstowego projektu wybieramy właściwości
- Z drzewa po lewej stronie wybieramy *Java Build Path*, wybieramy *Add External JARs...*
- Wskazujemy plik *sqljdbc4.jar* i wszystko zatwierdzamy.

W wierszu poleceń: oglądamy run.bat i run.sh

Przykład

Przykład: Pierwszy

- PobierzDane.java
- PobierzDane2.java
- ModyfikujDane.java

Transakcje

Do zarządzania transakcjami mamy metody

- Connection.setAutoCommit(true|false)
- Connection.commit()
- Connection.rollback()

Obsługa punktów zapisu odbywa się za pomocą

- Obiektu Savepoint
- Metody Connection.setSavePoint(nazwa_punktu)
- Metody Connection.rollback(nazwa_punktu)

Transakcje

Najpierw, za pomocą poniższej konstrukcji sprawdzamy, czy DBMS obsługuje dany poziom izolacji:

```
DatabaseMetaData dbMd = connection.getMetaData();
if (dbMd.supportsTransactionIsolationLevel(poziom))
{...}
```

Potem możemy ustawić poziom izolacji za pomocą metody

- Connection.setTransactionIsolation

Aby określić poziom izolacji, mamy to dyspozycji stałe statyczne

- Connection.TRANSACTION_READ_COMMITTED
- Connection.TRANSACTION_READ_UNCOMMITTED
- Connection.TRANSACTION_REPEATABLE_READ
- Connection.TRANSACTION_SERIALIZABLE

Przykład

Przykład: Transakcje

- Transakcje.java
- SavePoints.java

Operowanie na wyniku

- Do tej pory tworzyliśmy „statement” za pomocą `Connection.createStatement()`
- Utworzone wyniki były kursorami tylko-do-odczytu i przewijalnymi tylko do przodu (mieliśmy tylko rs.next())
- Metoda createStatement() jest przeciążona, w związku z czym możemy ją wywoływać z parametrami
- Odpowiednie ustawienie tych parametrów umożliwi nam przewijanie kursora w dowolną stroną oraz także modyfikację danych związanych z kursorem

Operowanie na wyniku

Przyjrzyjmy się metodzie `createStatement()`

- Składnia:
  - `Connection.createStatement(int resultSetType, int resultSetConcurrency)`
- Parametry:
  - resultSetType - określa widoczność zmian innych transakcji
    - ResultSet.TYPE_FORWARD_ONLY: przewijanie tylko do przodu (domyślnie)
    - ResultSet.TYPE_SCROLL_INSENSITIVE: niewrażliwy na zmiany innych użytkowników
    - ResultSet.TYPE_SCROLL_SENSITIVE: wrażliwy na zmiany innych użytkowników
  - resultSetConcurrency - określa, czy kursor ma być read-only:
    - ResultSet.CONCUR_READ_ONLY: domyślne
    - ResultSet.CONCUR_UPDATABLE: pozwala na modyfikację danych związanych z kursorem

Operowanie na wyniku

Wybrane metody obiektu `ResultSet`

- rs.next(), rs.previous()
- rs.first(), rs.last(), rs.beforeFirst(), rs.afterLast()
- rs.isFirst(), rs.isLast(), rs.isBeforeFirst(), rs.isAfterLast()
- rs.absolute(liczba), rs.relative(liczba)
- rs.insertRow(), rs.deleteRow(), rs.updateRow()
- rs.cancelRowUpdates(), rs.refreshRow()
- rs.getXXX(), updateXXX("kolumna", wartosc)
- rs.moveToInsertRow(), rs.moveToCurrentRow()

Przykład

Przykład: UpdatableResultSet

- ManipulacjaDanymi.java

Preparowany SQL

- Kiedy używamy preparacji zapytań?
  - Treść zapytania pozostaje niezmienna
  - Zapytanie jest często przetwarzane
    - Zapytanie jest wysyłane do bazy i potem prekompilowane
- Jak używamy?
  - Zapytanie reprezentuje PreparedStatement
  - Tworzymy poprzez Connection.prepareStatement()
  - Parametry są reprezentowane przez ?
  - Mamy metody typu stmt.setXXX(nr_kol, wartość)

Wykonywanie procedur składowanych

- Do wywoływania procedur i funkcji używamy obiektów klasy CallableStatement
- Tworzymy je poprzez Connection.prepareCall()
- Funkcja prepareCall może przyjąć parametr w jednej z dwóch postaci:
  - `{?= call <procedure-name>[<arg1>,<arg2>, ...]}`
  - `{call <procedure-name>[<arg1>,<arg2>, ...]}`
- Parametry procedury są reprezentowane przez ?
- Parametry wejściowe i wyjściowe ustawiamy przez
  - `setXXX(String parameterName, XXX wartość)`
  - `registerOutParameter(int paramIndex, int sqlType)`
- Parametry są typu
  - IN — używamy setXXX()
  - OUT — używamy registerOutParameter()
  - IN OUT — używamy setXXX() i registerOutParameter()

Przykład

Przykład: PreparowanieWywolanie

- Preparowanie.java
- Wywolywanie.java

Wsady

- Co to jest wsad i do czego służy?
- Obsługa wsadów jest w każdym z obiektów Statement, PreparedStatement, CallableStatement
- W każdym obiekcie do obsługi wsadów mamy metody
  - void addBatch(String s)
  - void clearBatch()
  - int[] executeBatch()
- Dostępny jest wyjątek BatchUpdateException oraz jego metoda getUpdateCounts()
- Należy pamiętać, że we wsadach mogą być procedury tylko o parametrach typu IN

Przykład

Przykład: Wsady

- Wsady.java
- Wsady2.java
- Wsady3.java

Metadane

- Z wynikiem nie są dostarczane informacje o strukturze tych danych (dlaczego?)
- Dostęp do informacji o strukturze jest poprzez interfejs ResulSetMetaData, który pozwoli ustalić
  - Ile jest kolumn w wyniku?
  - Czy dopuszczane są wartości NULL?
  - Jakie są etykiety nagłówków kolumn?
  - Jakie są nazwy kolumn?
  - Jakie są typy kolumn?
  - Jaka jest tabela źródłowa dla kolumn?
- Informacje o bazie danych otrzymamy poprzez interfejs DatabaseMetaData
  - Informacje o nazwie DBMS, wersji
  - Informacje o sterowniku, nazwa użytkownika
- Informacji o parametrach PreparedStatement dostarcza interfejs ParameterMetaData
  - Liczba parametrów

Przykład

Przykład: Metadane

- Metadane.java

HSQLDB

Wprowadzenie

- Lekki DBMS napisany w Javie
- Do pobrania ze strony http://www.hsqldb.org/
- W pobranym pakiecie jest biblioteka i są przykłady

HSQLDB

Jak tego używać?

- Rozpakujemy pobranego zip'a
- Uruchamiamy serwer odpalając skrypt `demo/runServer.bat`
- Uruchamiamy narzędzie do zarządzania odpalając skrypt `demo/runManagerSwing.bat`
- W narzędziu *HSQL Database Manager* łaczymy się bazą korzystając z następujących parametrów:
  - Type: *HSQL Database Engine Standalone*
  - Driver: *org.hsqldb.jdbcDriver*
  - URL: *jdbc:hsqldb:file:/sciezka/do/pliku*
  - User: *sa*
  - Password:
- W kodzie Javy z bazą łączymy się poprzez wpisanie:

```
Class.forName("org.hsqldb.jdbcDriver");
connection = DriverManager.getConnection("jdbc:hsqldb:hsqldbtest.db", "sa", "");
```