Wraz z powstaniem nowego, przenośnego, niezależnego od platformy i architektury języka programowania pojawiła się idea skonstruowania nowego interfejsu bazodanowego, który spełniałby podobne wymagania jak sam język. Naturalną konsekwencją takiego rozumowania było oderwanie się od już istniejących API, w szczególności wprowadzonego przez Microsoft standardu Open DataBase Connectivity (ODBC), i opracowanie nowej technologii, wykonanej w pełni w Javie.

Java DataBase Connectivity (JDBC) jest interfejsem programistycznym niskiego poziomu wywołującym bezpośrednio polecenia języka SQL, skonstruowanym w Javie i przeznaczonym dla języka Java. Interfejs JDBC umożliwia ustandaryzowany dostęp do większości obecnych na rynku systemów bazodanowych.

Dlaczego JDBC jest lepszy niż ODBC?
Stworzenie nowego interfejsu dostępu do systemu zarządzania bazą danych podyktowane było głównie faktem, iż korzystanie z interfejsu firmy Microsoft zaadaptowanego dla programów Javy nie stanowiło dobrego rozwiązania. Główny problem stanowi fakt, że ODBC jest technologią stworzoną w C i dla systemów opartych o ten właśnie język. Konwersja ODBC na język Java jest praktycznie niemożliwa do zrealizowani z powodów konstrukcyjnych – np. problem wskaźników. JDBC zgodnie z zamysłem twórców jest również dużo prostszy w użyciu a jednocześnie bardziej funkcjonalny niż ODBC. Nie wymaga ręcznych ustawień na platformach klientów, ponieważ kod JDBC jest automatycznie wykonywany na wszystkich platformach Javy od komputerów sieciowych do superkomputerów.

Dużą zaletą JDBC jest podobieństwo do technologii ODBC, dzięki czemu może być ona wykorzystywana przez programistów pracujących wcześniej z ODBC . Co ważne w czasie wprowadzania języka Java oraz sterowników JDBC prowadzono mosty JDBC-ODBC, dla baz danych nie posiadających „czystych" sterowników JDBC.

Zgodność ze standardami SQL.
JDBC określa pewien poziom zgodności z dotychczasowymi standardami SQL. Głównym założeniem twórców było aby każdy sterownik JDBC odpowiadał co najmniej wersji ANSI SQL-92 standardu SQL. Standard JDBC określa także, że zapytania SQL przesyłane są do odpowiedniego SZBD bez względu na możliwość ich realizacji. W przypadku, gdy dany SZBD nie potrafi obsłużyć zlecenia przekazanego przez JDBC, aplikacja podnosi określony wyjątek.
JDBC udostępnia również informacje o SZBD i jego cechach szczególnych (ustawieniach, możliwościach itd.). Informacje te przekazywane są użytkownikowi w postaci tzw. metadanych.

Sterowniki.
Sterownik JDBC (JDBC driver) jest zbiorem skompilowanych klas (bajtkod Javy), które implementują wszystkie interfejsy zawarte w java.sql oraz przedefiniowują pozostające tam klasy. W większości przypadków napisane są wyłącznie w Javie.
Implementacja bezpośrednio zależy od systemu zarządzania bazą danych, z którą będzie się łączyć aplikacja wykorzystująca sterownik, dlatego każdy sterownik JDBC służy do komunikacji z konkretną bazą danych i nie jest wykorzystywany w przypadku baz innych producentów.
Należy wspomnieć, iż ponieważ Java jest językiem w pełni przenośnym, producenci sterowników muszą przygotowywać tylko jeden pakiet dla każdej wersji standardu JDBC (za

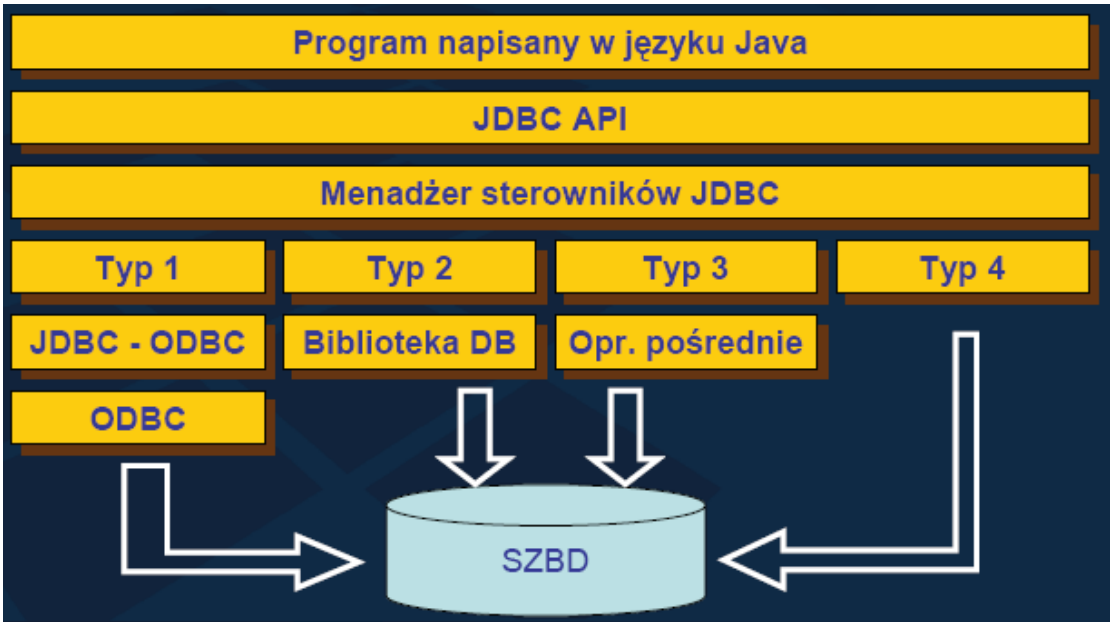
wyjątkiem sterowników 1. i 2. rodzaju). Tym samym, w odróżnieniu np. od interfejsów dla języka C++, z jednego binarium korzystamy np. w sytemie Linux i w Windows NT.

Większość sterowników JDBC jest darmowa i dostępna jako uzupełnienie pakietu JDBC, wystarczy je pobrać ze strony dostawców SZDB lub z repozytorium Sun'a.

Rozróżniamy cztery podstawowe typy sterowników JDBC:

1. Mosty JDBC-ODBC (JDBC-ODBC bridge). W tym wypadku korzystamy ze sterownika ODBC z którym komunikuje się most. Zapytania formułowane w Javie są tłumaczone na język sterownika ODBC, który odpowiada za wszelka komunikacja z bazą danych. Stosując to rozwiązanie należy się liczyć z opłatami za oprogramowanie ODBC.
2. Java do API SZBD (*Native-API partly-Java*). Program w Javie komunikuje się z interfejsem SZBD, a nie jak poprzednio, ze sterownikiem ODBC.
3. Pośrednie JDBC (*JDBC-Net pure Java*) - sterownik JDBC komunikuje się z serwerem pośredniczącym, za pomocą protokołu niezależnego od SZBD. Serwer tłumaczy polecenia na protokół konkretnego SZBD i przesyła do niego otrzymane zapytania. Serwer może pośredniczyć w wymianie danych pomiędzy wieloma klientami i wieloma *różnymi* SZBD. Tym samym jest to najbardziej ogólne i heterogeniczne rozwiązanie, obciążone względami bezpieczeństwa.
4. Bezpośrednie JDBC (*Native-protocol pure Java*) - sterownik JDBC komunikuje się bezpośrednio z bazą danych przy pomocy jej protokołu sieciowego (rozwiązanie najbardziej ogólne - stosowane często w sieciach wewnętrznych).

Model warstwowy opisanych rozwiązań:



Dwa pierwsze rozwiązania nie są sprzętowo niezależne, wymagają bowiem dostarczenia oprogramowania dedykowanego konkretnej platformie (sterownik ODBC (1) lub API bazy danych (2)).

Nawiązywanie połączenia
Aby skorzystać z klas java.sql, należy pobrać odpowiedni pakiet zawierający dany sterownik JDBC, a następnie umieścić go (zgodnie z hierarchią katalogów zawartą w archiwum) w katalogu widocznym dla maszyny wirtualnej Javy (zmienna CLASSPATH), dzięki czemu każda aplikacja będzie mogła go dynamicznie załadować. Do tego celu służy polecenie:

```
Class.forName("pełna_nazwa_sterownika")
```

W ten sposób można rejestrować dowolną liczbę sterowników.
Inną metodą na załadowanie sterownika jest ustalenie własności jdbc.drivers na nazwę ładowanej klasy:

```
java -Djdbc.drivers="oracle.jdbc.driver.OracleDriver" program
```

Jeżeli podana nazwa klasy sterownika jest niepoprawna, zgłaszany jest wyjątek „ClassNotFoundException”.

Klasy sterowników ("pełna_nazwa_sterownika") dla różnych SZBD:

1. dla bazy Oracle: oracle.jdbc.driver.OracleDriver
2. dla bazy IBM DB2:
   COM.ibm.db2.jdbc.app.DB2Driver (Typ 2),
   com.ibm.db2.jcc.DB2Driver (Typ 4)
3. dla bazy MS/SQL Server/Sybase:
   com.microsoft.sqlserver.jdbc.SQLServerDriver dla wersji 2005 MS/SQL Server (dla sterownika Microsoftu)
   com.microsoft.jdbc.sqlserver.SQLServerDriver dla wersji 2000 MS/SQL Server (dla sterownika Microsoftu)
4. dla bazy mySQL: com.mysql.jdbc.Driver
5. dla bazy PostgreSQL: org.postgresql.Driver

Po zarejestrowaniu sterownika możliwe jest nawiązanie połączenia z SZBD. Połączenie tworzone jest za pomocą statycznej metody „getConnection” klasy „DriverManager”. Pierwszym parametrem tej metody jest URL połączenia z SZBD, określający z jakim SZBD należy nawiązać połączenie oraz jego adres sieciowy. Kolejnymi parametrami są odpowiednio: nazwa użytkownika i hasło.

```
Connection con = DriverManager.getConnection(“adres_url”,”nazwa_urzytkownika”,
“hasło_urzytkownika”);
```

Format adresu URL może być dość skomplikowany i zawierać wiele różnych opcji połączenia. Przykładowe adresy URL, dla różnych SZBD (w najprostszej wersji):

1. IBM DB2:
   jdbc:db2//server:port/database (Typ 4 sterownika)
   jdbc:db2//server/database (Typ 4 sterownika)
   jdbc:db2:database (Typ 2 sterownika)
2. MS/SQL Server/Sybase:
   jdbc:sqlserver://server:port;databasename=database (MS/SQL 2005, sterownik Microsoftu)
   jdbc:microsoft:sqlserver://server:port;databasename=database (MS/SQL 2000, sterownik Microsoftu)
3. mySQL:

jdbc:mysql://server:port/database

4. Oracle:
   jdbc:oracle:thin:@//server:port/service (Typ 4 sterownika)
   jdbc:oracle:thin:@server:port:sid (Typ 4 sterownika)
   jdbc:oracle:oci:@//server:port/service (Typ 2 sterownika)
   jdbc:oracle:oci:@server:port:sid (Typ 2 sterownika)
5. PostgreSQL:
   jdbc:postgresql://server:port/database

server – adres serwera
port – port na którym nasłuchuje serwer
database – nazwa bazy danych

Metoda getConnection zwraca obiekt klasy implementującej interfejs „Connection", który reprezentuje nawiązane połączenie. Jeżeli nawiązanie połączenia nie powiedzie się, zostanie zgłoszony wyjątek „SQLException".

```
import java.sql.*;
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection(
 "jdbc:oracle:thin:@dblab.cs.put.poznan.pl:1521:dblab10g",
 "elearning_user", "elearning_pass");
con.close();
```

Zapytania
Klasa Statement odpowiada za przesyłanie wszelkich zleceń – operacji SQL do bazy danych. Obiekt klasy Statement tworzony jest w oparciu o wcześniej ustanowione połączenie z bazą danych. Statement nie posiada samodzielnego konstruktora, nowy obiekt jest zwracany przez metodę createStatement():

```
Connection con = DriverManager.getConnection (...)
Statement stmt = con.createStatement();
```

Wszystkie operacje związane z bazą danych wykonuje się poprzez utworzony obiekt (można utworzyć więcej niż jeden obiekt klasy Statement). Jest to jedyna metoda, by złożyć kolejne zapytanie podczas przeglądania wyników poprzedniego.
JDBC rozróżnia trzy odmienne kategorie zleceń do bazy danych. Kategorie te reprezentuje klasa Statement i jej dwie podklasy: PreparedStatement oraz CallableStatement.

Statement
Do wykonywania operacji na bazie danych służą bezpośrednio trzy metody klasy:

- executeQuery(query)
  używana jest do składania zwykłych zapytań SQL rozpoczynających się od słowa SELECT. W wyniku wykonania metody otrzymujemy listę wierszy opakowaną w obiekt klasy ResultSet (nigdy nie otrzymamy null).

- executeUpdate(query)
  używana jest do wykonywania operacji: insert, update i delete, a także operacji DDL (Data Definition Language): create table, drop table, czy też alter table. W wyniku zwraca pojedynczą liczbę, określającą liczbę wierszy tabeli, której dotyczyło zapytanie.
- execute(query)
  używana jest rzadko, wszędzie tam, gdzie w wyniku otrzymujemy więcej niż jedną listę wierszy lub więcej niż jedną liczbę zmodyfikowanych wierszy

query - treść (String) zapytania – dowolna, poprawna składniowo konstrukcja języka SQL

Trzeba pamiętać, że wykonanie powyższych operacji spowoduje zamknięcie obiektu ResultSet – otwartego w trakcie analizy zapytania. Pominięcie tego faktu, prowadzi do wielu błędów. Jeżeli w trakcie obróbki wyniku zapytania chcemy wykonywać kolejne operacje na bazie danych, należy utworzyć osobny obiekt Statement i z nim skojarzyć następne zapytanie.

ResultSet
Klasa ResultSet reprezentuje podstawową strukturę danych wynikowych dla zapytań SQL. Udostępnia ona metody pozwalające na przetwarzanie otrzymanych danych. Obiekt klasy ResultSet otrzymujemy w wyniku wykonania metody: executeQuery(). Aby uzyskać wartość i – tej kolumny danego wiersza stosujemy metodę getTyp (i) (Typ –typ wartości np. Int). Do następnego wiersza przesuwamy się korzystając z metody next(), uprzednio sprawdziwszy, czy następny wiersz istnieje (hasNext()).

Przykładowy schemat analizy wyników zapytania:

```
Statement stmt = con.createStatement() ;
ResultSet rs=stmt.executeQuery(
  "SELECT nazwisko,placa_pod FROM pracownicy");
while (rs.next()) {
  String nazwisko=rs.getString("NAZWISKO");
  float placa=rs.getFloat(2);
  System.out.println(nazwisko+" "+placa);
}
rs.close();
stmt.close();
```

Jeżeli pracujemy w trybie autocommit (transakcyjność), zapytanie zostaje zatwierdzone dopiero po wykonaniu całej transakcji i zamknięciu obiektu ResultSet (close()). Ma to duże znaczenie w przypadku operacji insert/update/delete.

Po liście wyników można się poruszać tylko w jednym kierunku – do przodu. Aby umożliwić elastyczne poruszanie się po liście wierszy wprowadzono następujące rozszerzenia klasy ResultSet:

ScrollableResultSet
Klasa umożliwia elastyczne poruszanie się po otrzymanej liście wierszy oraz pozwala na modyfikacje tabeli w trakcie przeglądania wyników zapytania.

O rodzaju zwracanego obiektu (wybór pomiędzy klasą ResultSet a ScrollableResultSet) decyduje metoda tworząca kontener Statement. Użycie bezparametrowego wywołania createStatement() jest rozwiązaniem standardowym. Natomiast zastosowanie createStatement (int jaki_rezultat, int jaka_współbieżność) tworzy kontener zdolny do tworzenia elastycznych wyników. Pierwszy argument decyduje o możliwości dwukierunkowego przewijania kursora oraz reakcji na zmiany dokonane w bazie danych po pobraniu danych. Dopuszczalne są trzy stałe:

- TYPE_FORWARD_ONLY - brak możliwości cofania kursora (podstawowa)
- TYPE_SCROLL_INSENSITIVE - dowolność w poruszaniu się po kursorze, włącznie z przemieszczaniem do dowolnego wiersza. Zmiany zaistniałe w bazie danych po wykonaniu zapytania nie wpływają na zawartość kursora.
- TYPE_SCROLL_SENSITIVE - poruszamy się jak w poprzednim przypadku, dodatkowo zmiany w bazie danych są odnotowywane przez kursor (najbardziej elastyczne rozwiązanie).

Drugi argument metody określa poziom współbieżności (możliwe są dwie stałe):

- CONCUR_READ_ONLY - działa w parze z opcją TYPE_FORWARD_ONLY (podstawowa)
- CONCUR_UPDATABLE - pozwala użytkownikowi na wykonywanie tzw. programowych operacji zapisu/modyfikacji/usunięcia danych znajdujących się bezpośrednio w bazie danych. Pozwala m.in. na obudowywanie kursorów warstwą prezentacyjną (formatka).

PreparedStatement

Dla usprawnienia procesu przesyłania danych wprowadzono mechanizm wstępnego przetwarzania zapytań. Zapytanie, po przesyłaniu do bazy danych, jest prekompilowane i od tego momentu użytkownik nie musi go przesyłać po raz kolejny. Zastosowania tego mechanizmu ograniczają się do wąskiego zakresu przypadków:

- treść zapytania nie zmienia się w czasie - zapytanie jest wielokrotnie ponawiane (procedura bezparametrowa)
- zapytanie jest przetwarzane bardzo często, jego treść można rozsądnie sparametryzować

W obu przypadkach można użyć osadzonych w SZBD, prekompilowanych zapytań.

Użycie zapytań prekompilowanych poleceń do aktualizacji danych:

```
PreparedStatement stmt = con.prepareStatement(
  "UPDATE pracownicy SET placa_pod= ?, etat = ?
   WHERE id prac= ?");
stmt.setFloat(1,2000);
stmt.setString(2,"PROFESOR");
stmt.setInt(3,140);
int changes =stmt.executeUpdate();
System.out.println("Zmodyfikowano "+changes+
  " krotek");
stmt.close();
```

Procedury

Sekwencje rozszerzające

JDBC umożliwia korzystanie z funkcji skalarnych wbudowanych w SZBD. Sterowniki spełniające warunki zgodności z JDBC muszą zapewniać poprawność działania z funkcjami skalarnymi, o ile odpowiadający im SZBD funkcje te udostępnia.
Z uwagi na różnorodność wywołań funkcji u różnych producentów baz danych, JDBC wprowadziło tzw. sekwencje rozszerzające (escape sequences) - składniowe ramy dla wywołań funkcji i procedur składowanych, przekazywania daty, czasu itp.
Wywołanie funkcji opatrzone jest następującą klauzulą:

{fn <nazwa_funkcji ()>}

np. SELECT {fn concat (string, "napis")} FROM...

Sekwencje rozszerzające określają także sposób wywołania i przekazywania parametrów dla procedur składowanych.

Procedury składowane

Poza funkcjami skalarnymi, JDBC umożliwia wykorzystanie procedur składowanych (stored procedures) - procedur i funkcji zgromadzonych po stronie SZBD. Wołanie procedur oraz przekazywanie parametrów (w obu kierunkach) określają reguły składniowe sekwencji rozszerzających. Nośnikiem dla zapytań zawierających wywołania procedur są obiekty klasy CallableStatement, stanowiącej rozszerzenie klasy Statement. Po utworzeniu obiektu wspomnianej klasy należy przygotować treść zapytania:

? = call nazwa_procedury (? ?)

Znaki zapytania odpowiadają zmiennym przekazywanym procedurze. JDBC dopuszcza zmienne trzech typów (koncepcyjnych):

- IN - wartości zmiennych przekazywane są procedurze.
- OUT - z argumentem wiążemy zmienną programu - po wykonaniu procedury zmienna będzie miała odpowiednio zmodyfikowaną wartość.
- INOUT - argument spełnia obie powyższe role.

Przykład:

```
CallableStatement cs = con.prepareCall (" {? = call nazwa_procedury (? ?)} ");

cs.registerParameterOut (1, Types.INTEGER);
cs.setString (2, "arg_napisowy");
cs.setString (3, "2_arg_napisowy");
cs.registerParameterOut (3, Types.STRING);

cs.execute ();

int wynik = cs.getInt (1);
String arg_2 = cs.getString (3);
```

Operacje Grupowe

Wersja 2.0 JDBC wprowadza pojęcie operacji grupowych (batch updates). Umożliwiają one zestawienie wielu zapytań do bazy danych i wykonanie ich jednorazowo. Poszczególne

sterowniki JDBC nie muszą wspierać tego rozwiązania, ani zapewniać istotnej poprawy wydajności. Sprawdzenie, czy dany sterownik implementuje operacje grupowe odbywa się poprzez wywołanie metody:
DatabaseMetaData.supportsBatchUpdates();

Zarządzanie zapytaniami wchodzącymi w skład operacji grupowej odbywa się poprzez zestaw metod klasy Statement:

- addBatch("tresc_zapytania") - dodanie cząstkowego zapytania
- clearBatch() - usunięcie wszystkich zapytań wchodzących w skład jednej grupy
- executeBatch() - wykonanie zbiorowego zapytania - w wyniku otrzymujemy tablicę liczb całkowitych, które odpowiadają kodom wykonania każdego cząstkowego zapytania

Wystąpienie błędu podczas wykonania cząstkowego zapytania powoduje podniesienie wyjątku BatchUpdateException(). Jego obsługa wiąże się m.in. z analizą wynikowej tablicy kodów, dostępnej poprzez wywołanie metody getUpdateCounts().

Wyjątki, ostrzeżenia i skracanie danych
Większość metod udostępnianych przez JDBC zmusza użytkownika do obsługi wyjątków. Wyjątki te - obiekty klasy SQLException - opisują błędy zgłaszane przez bazę danych oraz sam sterownik JDBC.
JDBC udostępnia także mechanizm powiadamiania o sytuacjach, który nie kwalifikują się do miana błędu, choć informacja o ich zajściu może być kluczowa dla poprawnego działania aplikacji. Klasą opisującą wspomniany mechanizm jest SQLWarning. Warto pamiętać, że po każdej aktualizacji danych (wykonanie zapytania, pobranie wiersza itp.), informacja o wcześniejszych ostrzeżeniach jest bezpowrotnie tracona.

Jeżeli w wyniku wykonania operacji bazodanowej rozmiar zwracanych danych przekracza ustalone granice, może dojść do ich automatycznego skrócenia (data truncation). Informacja o obcięciu danych pobieranych z bazy jest składowana w postaci obiektu klasy DataTruncation, która z kolei jest podklasą SQLWarning. W przeciwieństwie do operacji odczytu, skrócenie danych podczas zapisywania informacji do bazy powoduje wygenerowania wyjątku.

Bibliografia:
http://stencel.mimuw.edu.pl/abwi/20011218.b.JDBC/
http://wazniak.mimuw.edu.pl/images/0/00/BD-1st-2.4-lab10.tresc-1.1-kolor.pdf