Rozdział 9 Obsługa baz danych w języku Java

9.1 Obsługa baz danych w Javie - pakiet SQL

Ze względu na tak dużą popularność baz danych oraz z uwagi na liczne mechanizmy pracy z bazami danych jakie dostarcza język Java warto omówić podstawy bibliteki Java JDBC API. JDBC jest znakiem towarowym firmy SUN (nie jest to skrót, często tłumaczony jako Java Database Connectivity) określającym interfejs języka Java przeznaczony do wykonywania poleceń SQL (Structured Query Language). Strukturalny język zapytań SQL jest uniwersalnym, znormalizowanym (Java obsługuje SQL zgodnie z normą ANSI/ISO/IEC 9075:1992, inaczej SQL-2) językiem pracy z bazami danych. W oparciu o SQL tworzone są systemy zarządzania bazami danych (DBMS - DataBase Management System), których rolą jest m.in tworzenie struktury bazy, wypełnianie bazy danych, usuwanie lub aktualizacja rekordów, nadawanie praw dostępu, itp. JDBC jest interfejsem niskiego poziomu wywołującym bezpośrednio polecenia języka SQL. Rolę JDBC można więc ująć w trzech punktach:

1. utworzenie połączenia z bazą danych,
2. wysłanie polecenia (poleceń) SQL,
3. przetworzenie otrzymanych wyników.

JDBC może stanowić podstawę do tworzenia interfejsów wyższego rzędu. Znane są prace nad stworzeniem interfejsu mieszającego elementy SQL i Javy (np. umieszczenie zmiennych Javy w SQL). Określony preprocesor wbudowanego w Javie języka SQL tłumaczyłby stworzone rozkazy na rozkazy niskiego poziomu zgodnie z JDBC. Inna wersja interfejsu wysokiego rzędu zakłada odwzorowanie tabel na klasy. Każdy rekord staje się wówczas obiektem danej klasy. Tworzenie interfejsów wyższego poziomu jest również istotne z punktu widzenia planowanego modelu dostępu do bazy danych. Popularna dwu-warstwowa metoda dostępu (two-tier) daje bezpośredni dostęp do bazy danych (aplikacja/applet - baza danych) . Oznacza to, że musimy znać format danych bazy by móc pobrać lub zmienić informacje. W przypadku bardziej uniwersalnym dodaje się trzecią warstwę w modelu dostępu do bazy (aplikacja/applet (GUI) - serwer - baza danych). W modelu trzy punktowym stosuje się interfejs wyższego poziomu po to aby struktura dostępu do bazy danych stanowiła pewną abstrakcję, co umożliwia tworzenie różnych klientów bez potrzeby zagłębiania się w szczegóły protokołów wymiany danych z bazą. Tak stworzona konstrukcja dostępu do bazy danych uwalnia klienta od znajomości organizacji bazy danych, co za tym idzie możliwe są prawie dowolne modyfikacje ustawień bazy danych np. kilka rozproszonych baz zamiast jednej.
Istnieją inne interfejsy dostępu do baz danych jak na przykład popularny Open DataBase Connectivity firmy Microsoft. Jednak korzystanie z ODBC przez programy Javy nie stanowi dobrego rozwiązania ponieważ:

1. występuje różnica języków programowania - ODBC jest stworzony w C, a co za tym idzie konieczna konwersja porzuca cechy języka Java, a często staje się nie realizowalna ze względu na inne koncepcje np. problem wskaźników
2. korzystanie z ODBC jest znacznie trudniejsze, a nauka pochłania zbyt dużo czasu,
3. praca z ODBC wymaga ręcznych ustawień na wszystkich platformach klientów, podczas gdy korzystanie z JDBC umożliwia automatyczne wykorzystanie kodu JDBC na wszystkich platformach Javy począwszy od komputerów sieciowych do superkomputerów.

W okresie wprowadzania języka Java oraz sterowników JDBC stworzono (JavaSoft) mosty JDBC-ODBC, będące rozwiązaniem dla tych, którzy korzystają z baz danych nie posiadających innych, "czystych" sterowników JDBC.

9.2 Utworzenie połączenia z bazą danych

Stworzenie połączenia z bazą danych polega utworzeniu obiektu Connection. W tym celu stosuje się jedną ze statycznych metod DriverManager.getConnection(). Każda metoda getConnection() zawiera jako argument adres URL dostępu do bazy danych. Adres ten definiowany jest poprzez trzy człony:

jdbc:<subprotocol>:<subname>.

Pierwszy element adresu jest stały i nosi nazwę jdbc. Określa on typ protokołu. Kolejny element stanowi nazwa sterownika lub mechanizmu połączenia do bazy danych. Przykładowo mogą to być nazwy: msql - sterownik dla bazy mSQL, odbc - mechanizm dla sterowników typu ODBC. Ostatna część adresu zawiera opis konieczny do zlokalizowania bazy danych. Element ten zależy od sterownika czy mechanizmu połączeń i może zawierać dodatkowe rozszerzenia zgodnie z koncepcją przyjętą przez twórcę sterownika. Standardowo omawiana część adresu wygląda następująco:

//hostname:port/subsubname.

Przykładowe pełne adresy url mogą wyglądać następująco:

jdbc:odbc:biomed, jdbc:msql://athens.imaginary.com:4333/db_test.

Jedna z metod getConnection() umożliwia dodatkowo przesłanie nazwy użytkownika i hasła dostępu do bazy danych:

getConnection(String url, String user, String password).

Połączenie byłoby niemożliwe bez istnienia sterowników. Zarządzaniem sterownikami, które zarejestrowały się za pomocą metody DriverManager.registerDriver() zajmuje się klasa DriverManager (np. metody getDriver(), getDrivers()). Klasy sterowników powinny zawierać kod statyczny (static {}), który w wyniku ładowania tych klas stwarza obiekt danej klasy automatycznie rejestrującej się za pomocą metody DriverManager.registerDriver(). Ładowanie sterownika (a więc jego rejestracja) odbywa się najczęściej poprzez wykorzystanie

metody Class.forName(). Ta metoda ładowania sterownika nie zależy od ustawień zewnętrznych (konfiguracji sterowników) i ma charakter dynamiczny. Przykładowe ładowanie sterownika o nazwie "oracle.db.Driver" wykonane jest poprzez zastosowanie metody Class.forName("oracle.db.Driver"). Fragment kodu obrazujący etap łączenia się z bazą danych ukazano poniżej:

```
 String url = "jdbc:odbc:kurs";
// przykładowa nazwa drivera - slowo "kurs" jest nazwą zasobów
//definiowaną w ODBC dla bazy np. pliku tekstowego
String username = ""; //brak parametrów dostępu do pliku tekstowego
String password = "";
try {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   //ładowanie sterownika -  most JDBC-ODBC
} catch (Exception e) {
   System.out.println("Blad ladowania sterownika JDBC/ODBC.");
   return;
}
Connection c = null;
try {
   c = DriverManager.getConnection (url, username, password);  //połączenie
} catch (Exception e) {
   System.err.println("Wystapil problem z polaczeniem do "+url);
}
```

Powyższy przykład zakłada rejestrację w ODBC bazy tekstowej o dostępie "kurs" i wskazanie odpowiedniego katalogu.

9.3 Sterowniki

Sterowniki, a więc pakiety kodu zawierającego implementację deklarowanych klas i metod (obsługa dostępu do bazy, np. implementacja metod interfejsu ResultSet jak first() getInt(); itp.) są zazwyczaj dzielone na cztery grupy:

a. sterowniki JDBC odwzorowujące żądaną funkcjonalność na funkcjonalność sterowników Open Database Connectivity – ODBC. Sterowniki te oznaczanę są często jako mosty JDBC-ODBC. Sun standardowo dostarcza swoją wersję mostu.
b. Sterowniki JDBC odwzorowujące żądaną funkcjonalność na funkcjonalność dostępną poprzez sterowniki binarne danej bazy danych.
c. Sterowniki JDBC odwzorowujące żądaną funkcjonalność na funkcjonalność oprogramowania pośredniczącego w komunikacji z serwerem bazy danych, czyli z następuje tłumaczenie poleceń.
d. Sterowniki JDBC odwzorowujące żądaną funkcjonalność na funkcjonalność serwera bazy danych. Jest to w pełni zgodne z Javą rozwiązanie, które zapewnia najczęściej twórca oprogramowania serwera bazy danych.

Poniższy fragment kodu demonstruje zasady tworzenia połączenia i wykorzystywania kodu zdalnego (sterowniki) dla bazy danych „qabase” pracującej na serwerze Msql:

```
try{
```

```
        Class.forName("com.imaginary.sql.msql.MsqlDriver");
}catch (Exception e){
        System.out.println("Błąd wczytywania sterowników");
        return;
}
String URL = "jdbc:msql://biomed.eti.pg.gda.pl:1114/qabase";
String username ="msql";
String password="";
s=null;
con=null;
try{
        con=DriverManager.getConnection(URL,username,password);
        s=con.createStatement();
}catch (Exception e) {
        System.err.println("Błąd połączenia z "+URL);
}
```

W przypadku apletu sterowniki (pakiet kodu) musi być zainstalowany na serwerze WWW tam, skąd pochodzi aplet.

9.4 Wysłanie polecenia SQL

W celu wysłania polecenia SQL należy stworzyć obiekt Statement. Obiekt ten stanowi kontener dla wykonywanych poleceń SQL. Wykorzystywane są dodatkowo dwa kontenery: PreparedStatement oraz CallableStatement. Obiekt Statement jest wykorzystywany do wysyłania prostych poleceń SQL nie zawierających parametrów, obiekt PreparedStatement używany jest do wykonywania prekompilowanych (przygotowanych - prepared) poleceń SQL zawierających jedno lub więcej pól parametrów (oznaczanych znakiem "?"; tzw. parametry IN), natomiast obiekt CallableStatement jest wykorzystywany do stworzenia odwołania (call) do przechowywanych w bazie danych procedur. W celu stworzenia obiektu dla opisanych wyżej interfejsów wykorzystuje się trzy odpowiednie metody interfejsu Connection: createStatement() - dla interfejsu Statement, prepareStatement() - dla interfejsu PreparedStatement oraz prepareCall() dla interfejsu CallableStatement. Przykładowo fragment kodu tworzący obiekt wyrażenia Statement może wyglądać następująco:

```
Connection c = null;
    try {
       c = DriverManager.getConnection (url, username, password);  //połączenie
       Statement s = c.createStatement(); // tworzymy obiekt wyrażenia
    } catch (Exception e) {
       System.err.println("Wystapil problem z polaczeniem do "+url);
    }
```

Posiadając obiekt Statement można wykorzystać trzy podstawowe metody umożliwiające wykonanie polecenia SQL. Pierwsza z metod executeQuery() jest używana do wykonywania poleceń, których efekt daje pojedynczy zbiór rezultatów ResultSet np. wyrażenie SELECT - wybierz. Drugą metodę executeUpdate() wykorzystuje się przy wykonywaniu poleceń INSERT, UPDATE oraz DELETE a

także wyrażeń typu SQL DDL (Data Definition Language - język definicji danych) jak CREATE TABLE i DROP TABLE. Efekt działania pierwszych trzech poleceń daje modyfikację jednej lub więcej kolumn w zero i więcej wierszach tabeli. Zwracana wartość w wyniku działania metody executeUpdate() to liczba całkowita wskazująca ilość rzędów, które podlegały modyfikacjom. Dla wyrażeń SQL DDL zwracana wartość jest zawsze zero. Metoda execute() jest rzadko używana, ponieważ jest przygotowana do obsługi poleceń zwracających więcej niż jeden zbiór danych (więcej niż jeden obiekt ResultSet). Obsługa zwracanych danych jest więc kłopotliwa stąd metoda ta jest wykorzystywana dla obsługi specjalnych operacji. W przypadku obiektu PreparedStatement interfejs definiuje własne metody execute(), executeUpdate() oraz executeQuery(). Dlaczego? Czy nie wystarczy, że interfejs PreparedStatement dziedziczy wszystkie metody interfejsu Statement, a więc i te omawiane. Otóż nie. Obiekty Statement nie zawierają wyrażenia SQL, które musi być podane jako argument do ich metod. W przypadku obiektu PreparedStatement wyrażenie SQL musi być przygotowane – obiekt zawiera prekompilowane wyrażenie SQL. Dlatego wykonanie odpowiedniego polecenia polega na wywołaniu metody dla obiektu PreparedStatement bez podawania żadnego argumentu.
Poniżej przedstawiono porównanie wykonania polecenia SQL dla obiektu Statement oraz obiektu PreparedStatement:

Statement:
```
static String SQL =  "INSERT INTO kurs VALUES ('Mariusz', 'MK', 28)";
Statement s=...
s.executeUpdate(SQL);
```

PreparedStatement:
```
Connection c=...
PreparedStatement ps = c.prepareStatement("INSERT INTO kurs VALUES (?, ?, ?)");
        ps.setString(1,"Mariusz");
        ps.setString(2,"MK");
        ps.setInt(1, 28);
        ps.executeUpdate();
```

W pracy z zewnętrznymi procedurami przechowywanymi poza kodem programu istotą wykorzystania Javy jest stworzenie wyrażenia typu CallableStatement poprzez podanie jako parametru metody Connection.prepareCall() sekwencji ucieczki typu {call procedure_name[(?, ?)]} lub {? = call procedure_name[(?, ?)]} w przypadku gdy procedura zwraca wartość. Jak widać istota polega na znajomości nazwy i parametrów obsługiwanej procedury. Parametry ustawia się tak jak dla wyrażeń PreparedStatement natomiast pobiera się metodami getXXX(). Wykonanie polecenia (poleceń) procedury odbywa się poprzez wykorzystanie metody execute().

Posiadając wiedzę na temat stworzenia połączenia oraz przygotowywania i wykonywania wyrażeń można wykonać dwa przykładowe programy. Programy te wymagają ustawienia w ODBC systemu Win95/NT (Panel sterowania): dodanie ODBC typu plik tekstowy, nazwa udziału: kurs, katalog: c:\kurs\java. Ustawienia te są konieczne ponieważ wykorzystany zostanie pomost JDBC-ODBC jako sterownik do bazy danych. Obydwa programy generują to samo: bazę danych (pliki) o nazwie kurs

z przykładowymi danymi. Pierwszy program czyni to z pomocą wyrażeń Statement drugi z pomocą PreparedStatement.

Przykład 8.1:

```
//DBkurs.java:
import java.sql.*;

public class DBkurs {
   static String[] SQL = {
      "CREATE TABLE kurs ("+
         "uczestnik varchar (32),"+
         "inicjaly varchar (3),"+
         "wiek integer)",
      "INSERT INTO kurs VALUES ('Jarek', 'JS', 30)",
      "INSERT INTO kurs VALUES ('Andrzej', 'AM', 27)",
      "INSERT INTO kurs VALUES ('Ania', 'AM', 20)",
      "INSERT INTO kurs VALUES ('Marcin', 'MH', 25)",
   };
   public static void main(String[] args) {
      String url = "jdbc:odbc:kurs";
      String username = "";
      String password = "";
      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      } catch (Exception e) {
         System.out.println("Blad ladowania sterownika JDBC/ODBC.");
         return;
      }
      Statement s = null;
      Connection c = null;
      try {
         c = DriverManager.getConnection (url, username, password);
         s = c.createStatement();
      } catch (Exception e) {
         System.err.println("Wystapil problem z polaczeniem do "+url);
      }
      try {
         for (int i=0; i<SQL.length; i++) {
            s.executeUpdate(SQL[i]);
         }
         c.close();
      } catch (Exception e) {
         System.err.println("Wystapil problem z wyslaniem SQL do "+url+
            ": "+e.getMessage());
      }
   }
}// koniec public class DBkurs
```

Przykład 8.2:

```
//DBkurs2.java:
import java.sql.*;

public class DBkurs2 {
   static String SQL = "CREATE TABLE kurs ("+
```

```
            "uczestnik varchar (32),"+
            "inicjaly varchar (3),"+
            "wiek integer)";
    public static void main(String[] args) {
        String url = "jdbc:odbc:kurs";
        String username = "";
        String password = "";
         String[] imie= {"Jarek","Andrzej","Ania","Marcin"};
         String[] inic={"JS","AM", "AM", "MH"};
         int[] lat={30,27,20,25};
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        } catch (Exception e) {
            System.out.println("Blad ladowania sterownika JDBC/ODBC.");
            return;
        }
        Connection c = null;
         Statement s = null;
         PreparedStatement ps = null;
        try {
            c = DriverManager.getConnection (url,username,password);
            s = c.createStatement();
        } catch (Exception e) {
            System.err.println("Wystapil problem z polaczeniem do "+url);
        }
        try {
              s.executeUpdate(SQL);
              s.close();
               ps = c.prepareStatement("INSERT INTO kurs VALUES (?, ? , ?)");
               for(int n=0; n<4; n++){
                    ps.setObject(1,imie[n]);
                  //setObject zamiast setString z uwagi na problemy z konwersją typów danych do Text File ODBC (MS)
                    ps.setObject(2,inic[n]);
                    ps.setInt(3, lat[n]);
                    ps.executeUpdate();
               }
              ps.close();
        } catch (Exception e) {
            System.err.println("Wystapil problem z wyslaniem SQL do "+url+ ": "+e.getMessage());
        }
        finally{
          try{ c.close(); }
          catch(SQLException e) {
              e.printStackTrace();
          }
        }//od finally
    }
}// koniec public class DBkurs2
```

Dla potrzeb pracy z bazą danych za pomocą języka SQL istotne jest odwzorowanie typów danych Java->JDBC->SQL->DBMS. JDBC 2.0 API jest zgodny z SQL-2, a co więcej udostępnia typy danych zgodnych z propozycją standardu SQL-3 (np. typ danych BLOB). Teoretycznie twórca oprogramowania bazy danych powinien traktować typy danych z baz danych tak jakby to były typy danych SQL (i tak powinno

być). Oznacza to ponownie, że baza danych stanowi pewną abstrakcję. W pracy z wyrażeniami w JDBC istotne jest ustawianie parametrów wyjściowych, co powoduje wykorzystanie jednej z metod setXXX(); a także ważne jest ustawianie parametrów wejściowych (CallableStatement, ResultSet), co powoduje wykorzystanie jednej z metod getXXX(). W metodach pracy z danymi XXX oznacza typ danych. Poniższa tabela ukazuje przykładowe odwzorowanie typów pomiędzy Javą a JDBC(SQL) poprzez użycie metod getXXX().

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte	X	x	x	x	x	x	x	x	x	x	x	x	x						
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x						
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x						
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x						
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x						
getDouble	x	x	x	x	x	X	X	x	x	x	x	x	x						
getBigDecimal	x	x	x	x	x	x	x	X	X	x	x	x	x						
getBoolean	x	x	x	x	x	x	x	x	x	X	x	x	x						
getString	x	x	x	x	x	x	x	x	x	x	X	X	x	x	x	x	x	x	x
getBytes														X	X	x			
getDate											x	x	x				X		x
getTime											x	x	x					X	x
getTimestamp											x	x	x				x		X
getAsciiStream											x	x	X	x	x	x			
getUnicodeStream											x	x	X	x	x	x			
getBinaryStream														x	x	X			
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Rysunek 9.1 : Konwersja typów danych

Przy pracy z wyrażeniami SQL (a właściwie wykonywaniem tych wyrażeń) ważnym zagadnieniem są transakcje. Transakcja składa się z jednego lub więcej poleceń, które zostały wywołane, wykonane i potwierdzone (Commit) lub odrzucone (RollBack). Transakcja stanowi więc jednostkową operację. Zarządzanie transakcją

jest szczególnie ważne gdy chcemy wykonać naraz kilka operacji. Wówczas konieczne jest ustawienie pracy w stanie bez potwierdzania (Connection.setAutoCommit(false)- domyślnie ustawiona jest praca z automatycznym potwierdzaniem), a następnie po wykonaniu wyrażeń SQL wydawany jest rozkaz potwierdzenia (Connection.commit()). Potwierdzenie jest poleceniem, które powoduje, że zmiany spowodowane wyrażeniem SQL stają się stałe. W przypadku błędu (np. spowodowanego awarią sieci komputerowej) można wysłać polecenie przerwania (RollBack) powodujące odtworzenie stanu przed wykonaniem wyrażenia. Podsumowując można powiedzieć, że efekt działania każdego polecenia SQL jest tymczasowy do momentu wysłania polecenia potwierdzenia, kiedy to zmiany stają się stałe lub do momentu wysłania polecenia przerwania, kiedy to odtwarzany jest stan przed wykonaniem polecenia.

9.5 Rezultaty i ich przetwarzanie

W zależności od typu polecenia SQL, a co za tym idzie typu wyrażenia executeXXX(), możliwe są różne rezultaty wykonanych operacji. Metoda executeUpdate() zwraca liczbę zmienionych wierszy, metoda executeQuery() zwraca obiekt typu ResultSet zawierający wszystkie rekordy (wiersze) będące wynikiem wydania polecenia SQL (SELECT). Dostęp do danych zawartych w ResultSet następuje poprzez odpowiedni przesuw (ResultSet.next()) po rekordach (początkowo wskaźnik ustawiony jest przed pierwszym elementem) oraz odczyt wartości pól za pomocą metod podanych w powyższej tabelce typu getXXX(), gdzie identyfikator kolumny określany jest poprzez jej nazwę (typu String - ważna jest wielkość liter) lub numer kolejny w rekordzie (np. 1 kolumna, 2 kolumna, ...).
Informację o kolumnach obiektu ResultSet dostępne są poprzez wywołanie metod interfejsu ResultSetMetaData, którego obiekt zwracany jest poprzez przywołanie metody ResultSet.getMetaData. Przykładowe własności kolumn to nazwa kolumny- getColumnName(), czy typ kolumny- getColumnType(). Niektóre systemy zarządzania bazami danych umożliwiają tzw. pozycjonowane zmiany i kasowanie polegające na wprowadzaniu zmian w aktualnie dostępnym rekordzie w ResultSet. Możliwe jest wówczas wykorzystanie metod updateXXX() interfejsu ResultSet do aktualizacji danych. W celu zobrazowania techniki przetwarzania rezultatów wykonania polecenia SQL posłużmy się następującym przykładem:

Przykład 8.3:

```
//DBkurs3.java:

import java.sql.*;

public class DBkurs3 {


   public static void main(String[] args) {
      String url = "jdbc:odbc:kurs";
      String username = "";
      String password = "";
       String imie, inic;
       int lata;
```

```
      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      } catch (Exception e) {
         System.out.println("Blad ladowania sterownika JDBC/ODBC.");
         return;
      }

      Connection c = null;
       Statement s = null;
      try {
         c = DriverManager.getConnection (url,username,password);
         s = c.createStatement();
      } catch (Exception e) {
         System.err.println("Wystapil problem z polaczeniem do "+url);
      }
       try {
            ResultSet r=s.executeQuery("SELECT uczestnik, inicjaly, wiek FROM kurs");
            int n=0;
            while(r.next()){
                  n++;
                  imie=r.getString(1);  // w pętli pobieramy dane
                  inic=r.getString(2);
                  lata=r.getInt(3);
                  System.out.println("Dane rekordu nr: "+n+" to: "+imie+", "+inic+", "+lata);
            }
            s.close();
      } catch (Exception e) {
         System.err.println("Wystapil problem z wyslaniem SQL do "+url+ ": "+e.getMessage());
      }
      finally{
        try{ c.close(); }
        catch(SQLException e) {
            e.printStackTrace();
        }

       }//od finally
   }
}// koniec public class DBkurs3
```

W ten sposób pokazane zostały wszystkie trzy podstawowe zadania w pracy z bazami danych. Warto wspomnieć, że JDBC składa się faktycznie z dwóch API: jedna biblioteka standardowa dołączana do JDK (i tylko te elementy wykorzystano i omówiono powyżej) tzw. JDBC 2.0 core API oraz dodatkowa biblioteka będąca rozszerzeniem czyli JDBC 2.0 extension API. Tak ujęta budowa JDBC umożliwia ciągły rozwój biblioteki przy zachowaniu standardowych narzędzi dostarczanych z podstawowym środowiskiem Javy.