Podstawy otwartych języków programowania

Java Database Connectivity (JDBC)

Wiktor Wandachowicz

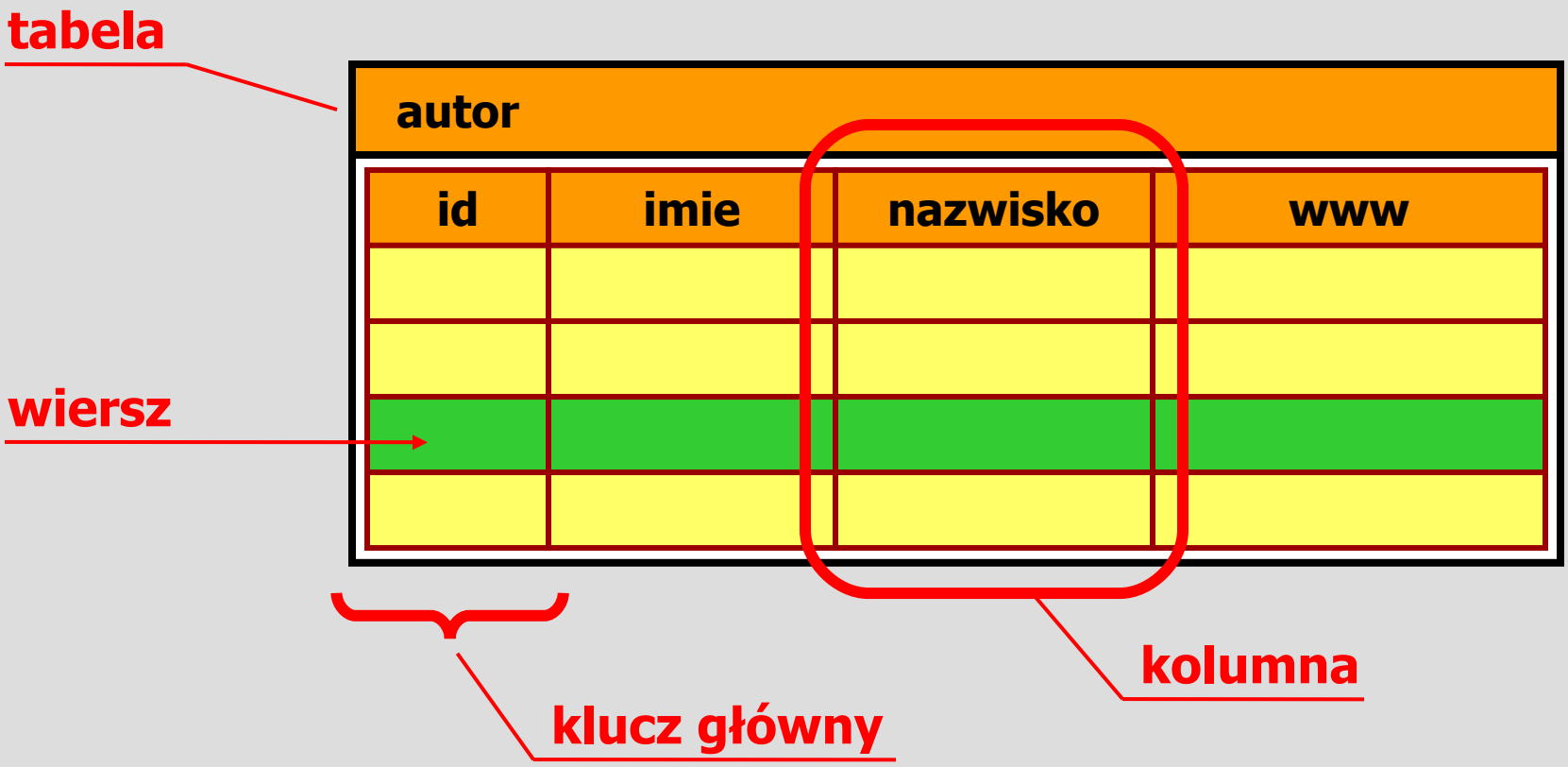
Treść wykładu

- Przypomnienie terminów bazodanowych
- Architektura JDBC
- Schemat działania przy dostępie do danych
- Analiza nazwy źródła danych podawanej przy nawiązywaniu połączenia
- Przykłady kodu

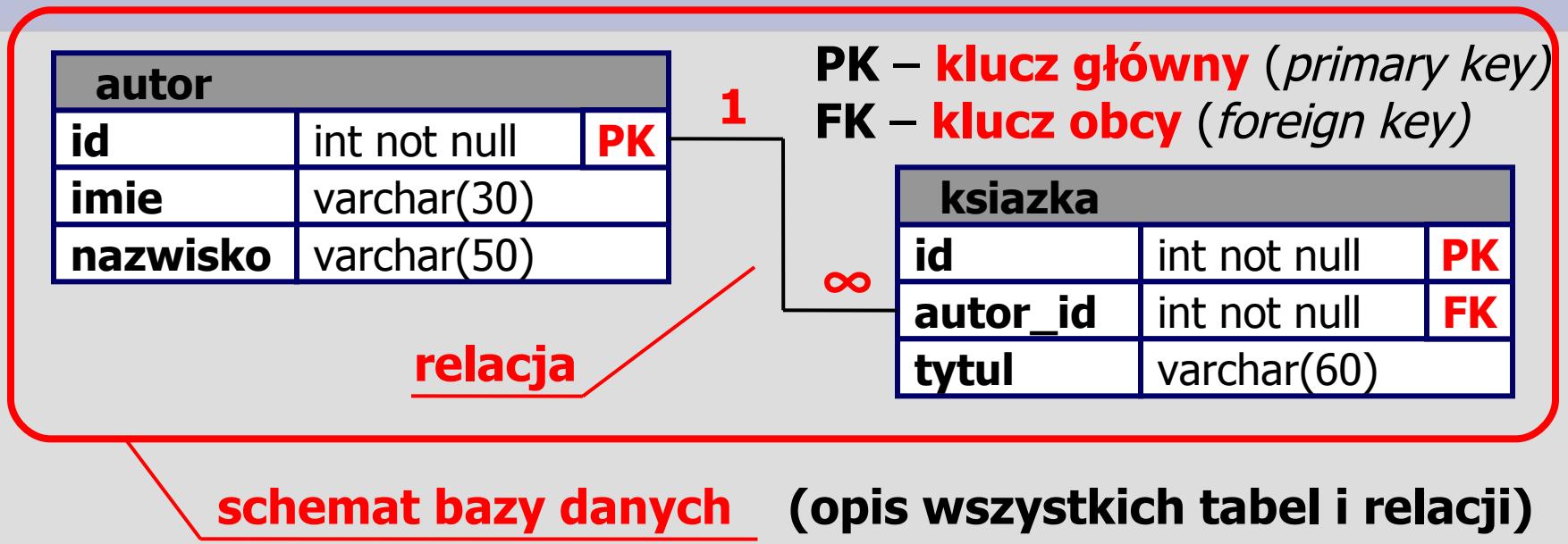
Terminy bazodanowe

- **Tabela** (relacja)
- **Wiersz** (krotka, rekord)
- **Kolumna** (atrybut)
- **Relacja** (związek)
- **Schemat**
- **Klucz, klucz główny, klucz obcy**

Terminy bazodanowe (przykład jednej tabeli)



Schemat bazy danych



autor

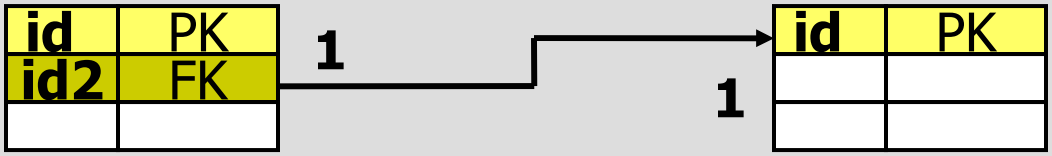
id	imie	nazwisko
1	Jeden	Taki
5	Frank	Herbert
6	Adam	Mickiewicz

ksiazka

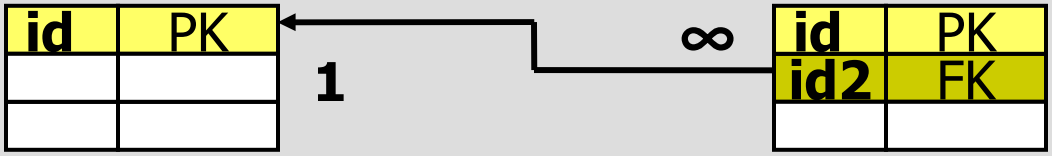
id	autor_id	tytul
1	1	Abc wędkarza
2	5	Diuna
3	6	Pan Tadeusz
4	5	Heretycy Diuny

Typy relacji

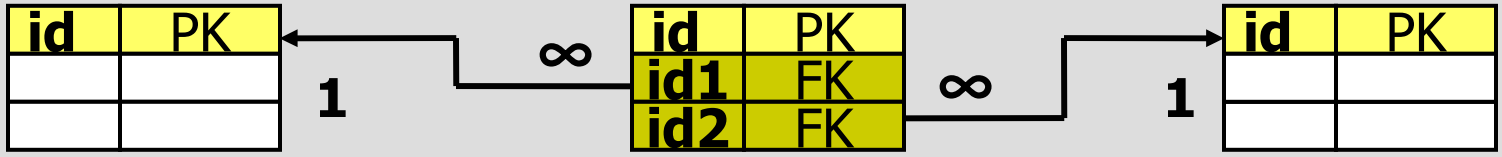
- **jeden do jednego** (**1:1**, *one to one*)



- **jeden do wielu** (**1:M**, *one to many*)



- **wiele do wielu** (**M:N**, *many to many*)

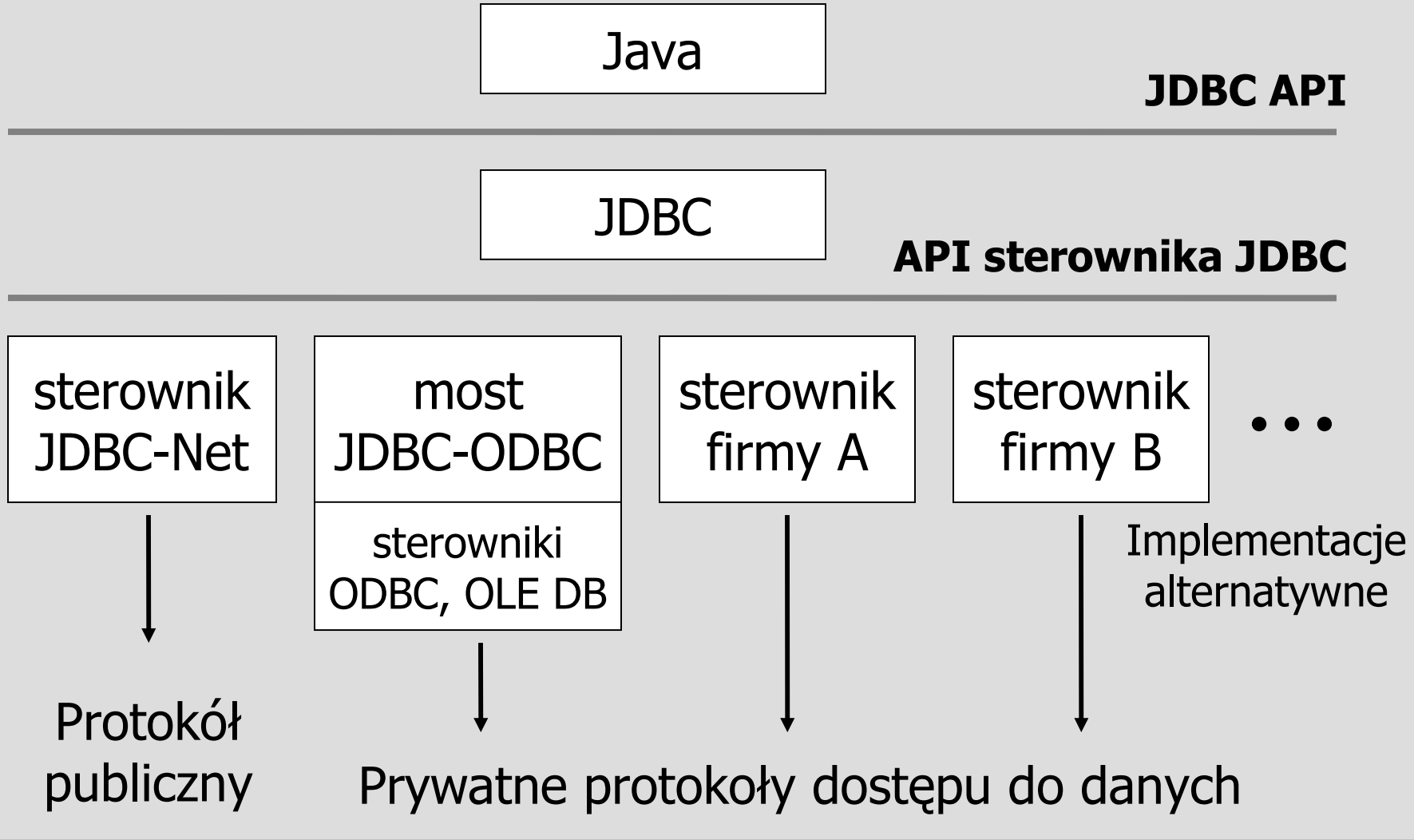


JDBC

JDBC™ API zapewnia uniwersalny dostęp do baz danych z poziomu języka Java.

- Niezależne od architektury, przenośne rozwiązanie
- Wystarczy sterownik do bazy zgodny z JDBC oraz odpowiednie nawiązanie połączenia (reszta programu się nie zmienia)
- Klasy JDBC są w pakiecie **java.sql**
- **JDBC 1.0** (tylko SQL), **JDBC 2.0** (manipulacje danymi), **JDBC 3.0** (nierelacyjne źródła danych)
- Wyjątki: **java.sql.SQLException**

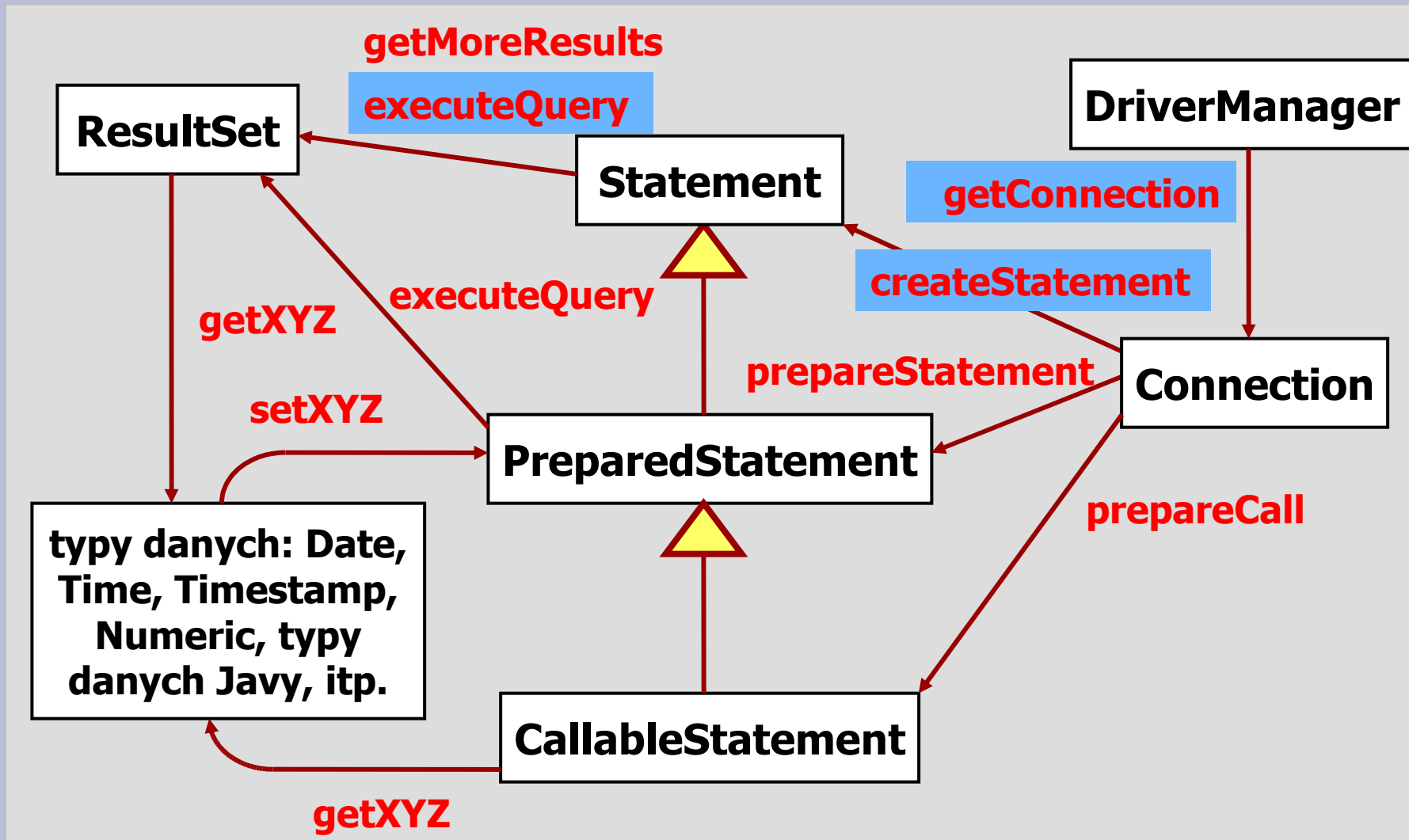
Architektura JDBC



Odczyt danych z bazy

1. Załadowanie sterownika
   - interfejs **java.sql.Driver**
2. Otwarcie połączenia
   - interfejs **java.sql.Connection**
3. Utworzenie obiektu do wykonywania poleceń
   - interfejs **java.sql.Statement**
4. Odczyt danych i zapamiętanie wyniku
   - interfejs **java.sql.ResultSet**

Schemat odczytu danych z bazy



1. Ładowanie sterownika

- Aby załadować sterownik trzeba użyć **Class.forName("nazwa.klasy.sterownika")**
- Maszyna wirtualna musi być w stanie odnaleźć klasę sterownika – trzeba podać pełną nazwę klasy, sama zaś klasa musi być dostępna przez CLASSPATH lub zapisana w ustawieniach maszyny wirtualnej
- Wszystkie sterowniki JDBC są zarządzane przez klasę **DriverManager**
- Każdy sterownik po załadowaniu sam rejestruje się przez wywołanie **DriverManager.registerDriver()** (i dlatego można go od tej pory używać)

Sterownik most JDBC-ODBC

- Wiele współczesnych baz danych ma sterowniki ODBC, jednak ODBC jest bardziej odpowiednie dla języka C niż Javy, jest też trudniejsze do nauczenia
- JDBC w porównaniu daje więcej możliwości, stanowi API wyższego poziomu – używa się kilku interfejsów składających się na JDBC, o resztę dba konkretny sterownik (są jednak też możliwości zaawansowane i rozszerzenia JDBC – pakiet **javax.sql**)
- Sterownik platformy Java, umożliwiający dostęp do istniejących sterowników ODBC:
  sun.jdbc.odbc.JdbcOdbcDriver

Sterowniki natywne

- Sterownik most korzysta z dodatkowej warstwy pośredniczącej (ODBC), można jednak mieć sterownik natywny – do konkretnej bazy danych
- Sterownik może być napisany całkowicie w Javie, jest wtedy przenośny i nie trzeba go specjalnie instalować na każdym komputerze (może być np. spakowany razem z aplikacją)
- Przykładowy sterownik do MySQL:
  com.mysql.jdbc.Driver
- Aby używać sterownika, można np. podać ścieżkę do archiwum przy starcie aplikacji:

```
java -cp .;mysql-connector-java-5.0.5-bin.jar Program
```

2. Nawiązywanie połączenia

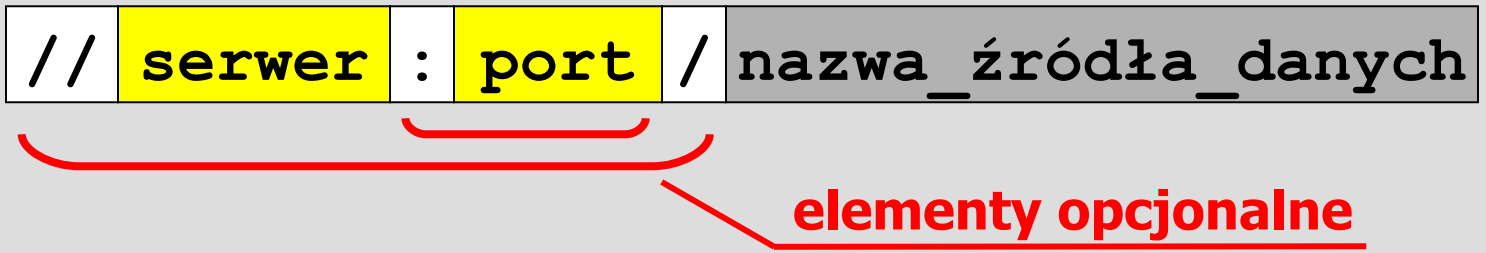
- JDBC 1.0 – **DriverManager.getConnection()**
- JDBC 3.0 – **DataSource.getConnection()**

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn;
try {
  conn = DriverManager.getConnection(
    "jdbc:odbc:biblioteka", "user", "pass");
} catch (SQLException e) {
  System.out.println(   "SQLException: " +
    e.getMessage()  + "\nSQLState:     " +
    e.getSQLState() + "\nVendorError:  " +
    e.getErrorCode());
  e.printStackTrace();
}
```

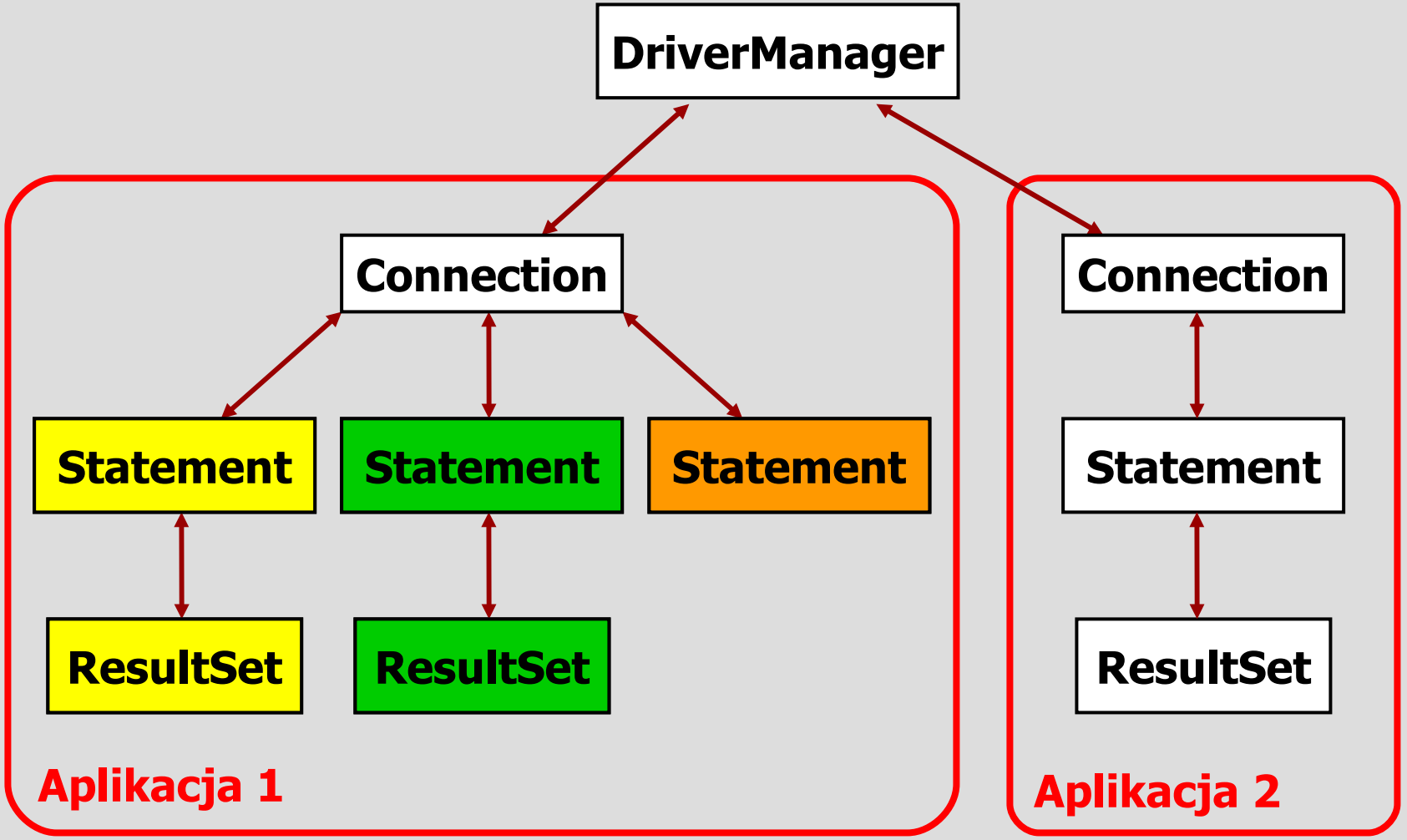
Identyfikator źródła danych

- Przy otwieraniu połączenia do bazy danych należy podać **łańcuch połączenia** (JDBC URL)
- Składa się on z następujących części:

- Podnazwa może być np. aliasem ODBC źródła danych (DSN – *Data Source Name*), albo może mieć ogólnie zalecaną postać:



Ile trzeba nam połączeń?



Stan połączeń

- Jeśli **DriverManager** zwrócił połączenie – to jest ono otwarte (jeśli nastąpi błąd, pojawi się wyjątek)
- Połączenie można zamknąć – **Connection.close()**
- Jeśli połączenie zostało zamknięte, metoda **Connection.isClosed()** zwraca true
- Nie używa się **isClosed()** aby sprawdzać czy połączenie jest w dobrym stanie (tylko czy zostało faktycznie zamknięte)
- Raczej można dowiedzieć się czy coś jest nie w porządku z połączeniem, łapiąc wyjątki jakie pojawiają się przy próbie wykonania operacji JDBC

3. Wydawanie poleceń

- Do wykonywania poleceń SQL (manipulowania danymi) w ramach połączenia z bazą danych, trzeba użyć **obiektu polecenia**.
- JDBC zawiera trzy klasy dla obiektów poleceń:
  - **Statement** – głównie odczyt danych, ale także wykonywanie zmian w bazie przez polecenia SQL
  - **PreparedStatement** – przechowuje wstępnie przygotowane polecenie SQL, z parametrami (szybsze przetwarzanie)
  - **CallableStatement** – wywołanie procedury składowanej z bazy danych

Tworzenie obiektów poleceń

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(
  "select * from autor");
stmt.executeUpdate(
  "insert into autor (imie, nazwisko) values " +
  " (\'Scott\',  \'McNealy\')," +
  " (\'Jonathan\', \'Schwartz\')");
```

```
PreparedStatement pstmt = conn.prepareStatement(
  "UPDATE EMPLOYEES SET SALARY = ? WHERE ID = ?");
pstmt.setBigDecimal(1, 153833.00);
pstmt.setInt(2, 110592);
```

```
CallableStatement cstmt = conn.prepareCall(
  "CALL GENERATE_REPORT");
```

4. Odczyt danych – ResultSet

- first() / last()
- beforeFirst() / afterLast()
- next() / previous()
- absolute() / relative()

przechodzenie po wierszach

- isFirst() / isLast()
- isBeforeFirst() / isAfterLast()

pytanie o położenie

Uwaga: wczytanie pustego zbioru danych powoduje, że w zbiorze nie ma pierwszego rekordu i cztery ostatnie metody zawsze zwracają wartość logiczną **false**

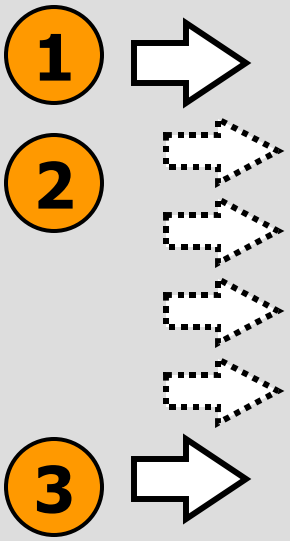
Różnice w obsłudze danych

- ResultSet w JDBC 1.0 pozwala odczytywać dane tylko do przodu
- ResultSet w JDBC 2.0 jest przewijalny, tzn. ma możliwość przechodzenia do przodu, do tyłu oraz do wybranego wiersza (ang. *scrollable*)
- W JDBC 2.0 można zmodyfikować dane odczytane w ResultSet używając jego metod, a następnie wysłać te zmiany do bazy

Przykład odczytu danych

```
/* Zakładamy, że połączenie 'conn' jest już
   utworzone i używa bazy 'biblioteka' */
Statement stmt = conn.createStatement();
// Odczyt zbioru danych
ResultSet rs = stmt.executeQuery(
  "select * from autor");
// Przechodzenie do kolejnych wierszy
while (rs.next()) {
  // Wyciągnięcie danych z bieżącego wiersza
  int    id       = rs.getInt(1);
  String imie     = rs.getString(2);
  String nazwisko = rs.getString("nazwisko");
  // Wyświetlenie danych
  System.out.println(id + " : " + imie +
    " , " + nazwisko);
}
```

Zmiany bieżącego wiersza



101	Adam	Nowak
108	Piotr	Goniec
152	Maria	Woźniczka
153	Stanisław	Wspaniały

- ResultSet na początku jest **przed** zbiorem danych (**isBeforeFirst()** zwraca **true**)
- Każde wywołanie **next()** przesuwa go do przodu o jeden (jeśli się udało – zwrócone będzie **true**)
- Po skończeniu się zbioru danych ResultSet jest **za** zbiorem danych (**isAfterLast()** zwraca **true**)

Porządki ?

- Należy zamykać połączenia i obiekty poleceń kiedy nie są potrzebne
- Służą do tego metody:
  Statement.close() oraz
  Connection.close()
- Zaleca się jawnie zamykać powyższe obiekty (szybsze zwalnianie zewnętrznych zasobów)
- Jeśli nie, połączenia będą zamykane dopiero przy odśmiecaniu pamięci

Typowe problemy

- W przypadku aplikacji przyjmujących dane przysyłane z przeglądarki internetowej niezmiernie ważną rzeczą jest upewnienie się czy przychodzące dane są poprawne.

- Do podstawowych elementów należą m.in.
  - walidacja Javascript po stronie przeglądarki
  - walidacja parametrów po stronie serwera
  - zapobieganie Cross-Site Scripting (XSS)
  - zapobieganie SQL Injection

Walidacja danych (sprawdzanie poprawności)

- Sprawdzanie poprawności powinno odbywać się w dwóch miejscach:
  - w przeglądarce (mniejsze obciążenie serwera)
  - w aplikacji na serwerze (zapobiega problemom, gdy Javascript nie działa w przeglądarce)
- W przypadku aplikacji JSP można wykorzystać znaczniki **`<jsp:useBean>`** oraz **`<jsp:setProperty property="*">`**

Cross-Site Scripting (XSS)

- Bez kontroli przysłanych danych może się okazać, że w bazie danych umieszczane są znaczniki HTML, które po prostym dołączeniu do strony powodują nieoczekiwane efekty (zmianę wyglądu, wyświetlanie zbędnych obrazków, itp.)
- Problem XSS objawia się tym, że w dodanymi znacznikami są `<script>` przez co przy ładowaniu lub wyświetlaniu strony przeglądarka wykonuje arbitralnie dodane programy Javascript.
- W celu zapobiegnięcia XSS jako minimum należy zamieniać znaki `<` na `<` i `>` na `>` przed zapisaniem danych w bazie.

SQL Injection

- Samodzielne budowanie wyrażeń SQL z ciągów znaków przysłanych z przeglądarki może powodować, że zostaną do nich dołączone arbitralnie podane, zbędne (niebezpieczne!) polecenia SQL.
- W celu zabezpieczenia się przed tymi problemami, należy zawsze przy podawaniu parametrów korzystać z obiektów **PreparedStatement** a nie **Statement**.

Przykład SQL Injection

- Zły kod:

```
String imie = "John";
String nazwisko = "K.\'); DROP DATABASE biblioteka;";
Statement stmt = conn.createStatement();
stmt.executeUpdate(
  "insert into autor (imie, nazwisko) values " +
  " (\'" + imie + "\', \'" + nazwisko + "\')");
```

- Sklejanie może doprowadzić tu do skasowania całej bazy danych.

Zapobieganie SQL Injection

- Dobry kod:

```
String imie = "John";
String nazwisko = "K.\'); DROP DATABASE biblioteka;";
PreparedStatement stmt = conn.prepareStatement(
  "insert into autor (imie, nazwisko) values " +
  " (?, ?)");
stmt.setString(1, imie);
stmt.setString(2, nazwisko);
stmt.executeUpdate();
```

- Wartości parametrów są podawane przez zbiór metod **setXyz()**.