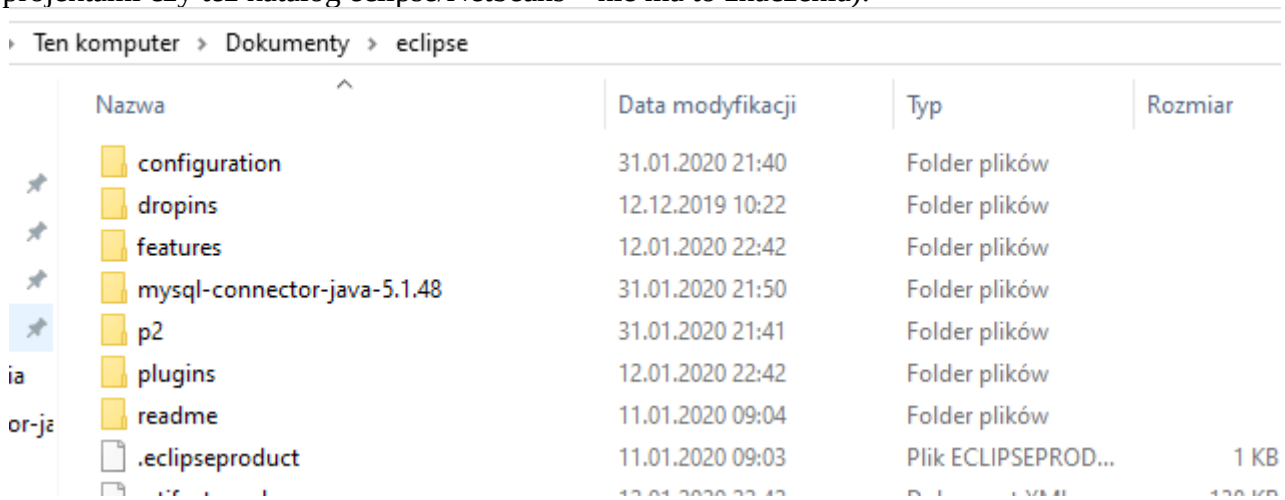
Java i połączenie do bazy danych MySQL

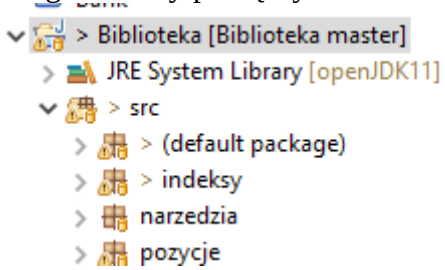
Język Java posiada wbudowaną obsługę języka SQL. Omawiany MySQL/MariaDB także można wykorzystać w swoich programach.
Przede wszystkim musimy posiadać odpowiedni sterownik do obsługi konkretnej bazy danych, gdyż JVM posiada jedynie abstrakcyjne klasy i metody obsługi połączeń (java.sql). Przykładowo sterownik do MySQL można pobrać ze strony:

https://dev.mysql.com/downloads/connector/j/5.1.html

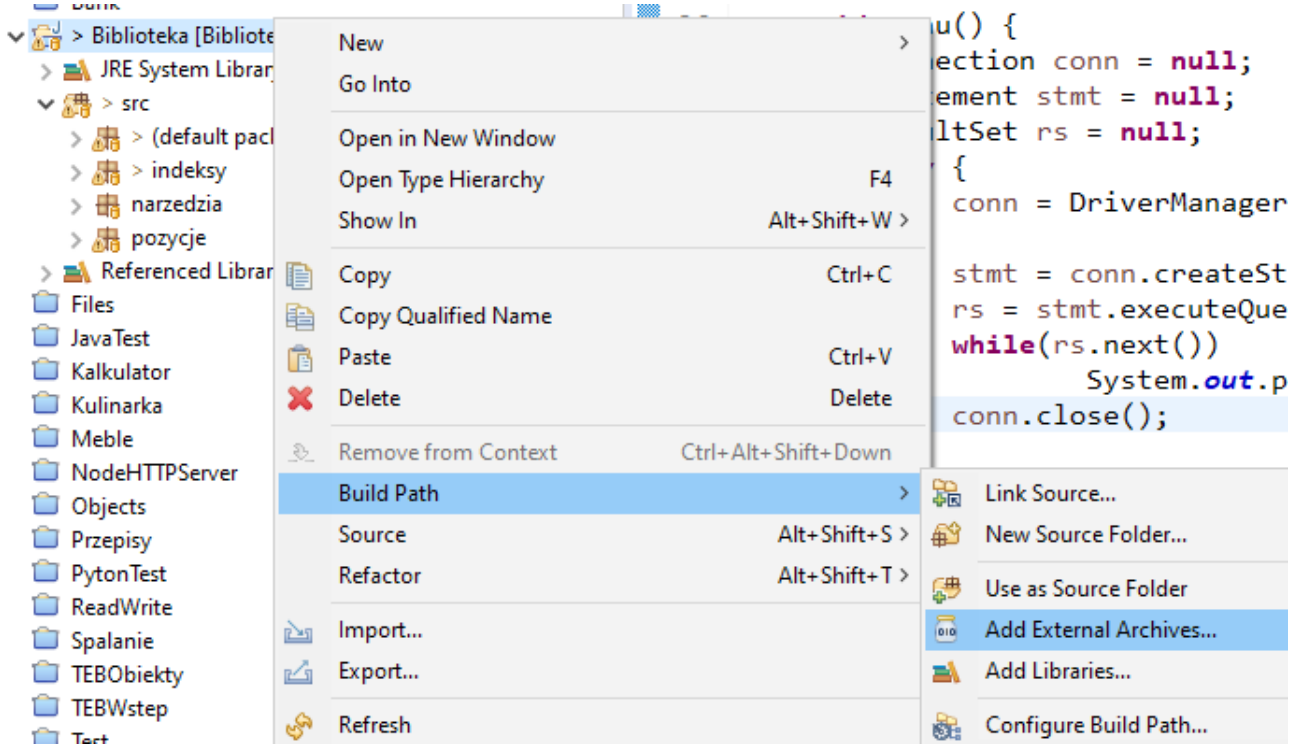
będzie to sterownik działający na dowolnej platformie sprzętowej i systemowej (działa w oparciu o JVM). W zależności od posiadanego programu do dekompresji archiwum możemy pobrać plik tar lub zip (mając 7zip można pobrać dowolny).
Mając już plik wypakowujemy go w dowolne miejsce na dysku (może to być np. katalog z naszymi projektami czy też katalog eclipse/Netbeans – nie ma to znaczenia):



Dokumentacja MySQL wspomina, że katalog z biblioteką połączenia wskazane jest podłączyć do ścieżki systemowej by każdy tworzony przez nas projekt miał do niej dostęp. Jednak dodatkowa ścieżka, szczególnie w przypadku systemu Windows, może spowalniać jego rozruch. Dlatego też poniżej zostanie pokazany inny sposób połączenia biblioteki do projektu:
1) odpalamy nasze środowisko. Zakładamy, że jest to Eclipse
2) Uruchamiamy projekt, do którego chcemy podłączyć



3) klikamy na projekcie prawym przyciskiem myszy; z menu wybieramy Build Path → Add External Archives…



Mając podłączoną bibliotekę możemy w dowolnej klasie naszego projektu rozpocząć pracę z bazą danych. Załóżymy, że posiadamy serwer bazy danych na lokalnym komputerze, wystawionym na standardowym porcie mysql (3306). Ponieważ do celu demonstracji użyty zostanie zestaw narzędziowy XAMPP, baza danych zostanie podłączona na koncie root (NIGDY NIE NALEŻY TEGO ROBIĆ W PROJEKCIE WDROŻONYM DO DZIAŁANIA) bez hasła (TYM BARDZIEJ). Instalacja i konfiguracja została omówiona wcześniej, opis instalacji wspomniany jest też w materiale http://planetatechnika.pl/SBD/phpmyadmin.pdf

W kodzie naszego projektu zaczynamy od dodania odpowiednich bibliotek:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
```

można też dodać:

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

jednak dodamy dodatkowe, zbędne nam klasy.

Następnie, gdzieś w kodzie, tworzymy trzy zmienne:

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
```

Pierwsza z nich odpowiada za utrzymywanie połączenia do bazy danych.
Druga odpowiada za komunikowanie się do poszczególnych tabel (operacje na nich → wykonywanie składni SQL).
Ostatnia zmienna przydatna jest jedynie w zapytaniach wybierających (SELECT) i pozwala na przechwytywanie danych z zapytania do programu.

Dalsze instrukcje należy wykonywać zawsze w otoczeniu try..catch ze względu na wiele wyjątków, jakie możne napotkać sterownik (np. brak komunikacji z bazą danych, brak wskazanego użytkownika, brak tabeli, błędne zapytanie, brak wyników itp.).

1) Połączenie z bazą danych:

```
 conn = DriverManager.getConnection("jdbc:mysql://localhost/biblioteka_java?
user=root&password=");
```

Połączenie w nowszych formacie (użytkownika i hasło można podać także jako osobne parametry funkcji). Należy pamiętać, że w sterowniku jdbc ZAWSZE musimy podać gospodarza (serwer) z bazą (w tym wypadku localhost, można podać dowolny adres URL, w tym adres IPv4/IPv6), można także podać port (jeżeli byłby inny niż domyślny). Przykładowo jeżeli chcielibyśmy połączyć się do bazy działającej na porcie 4000:

```
jdbc:mysql://localhost:4000/biblioteka_java?user=root&password=
```

lub działającej w lokalnej sieci na adresie 10.100.0.20:

```
jdbc:mysql://10.100.0.20/biblioteka_java?user=root&password=
```

Po nazwie/adresie serwera podajemy nazwę bazy danych, do której się łączymy. Z założenia musimy podać nazwę bazy danych (nie jest wskazane pracowanie na serwerze bez jednoznacznego wskazania bazy danych przeznaczonej dla programu). W naszym wypadku nazwa ta to biblioteka_java

Po znaku zapytania pojawiają się dodatkowe parametry połączenia. Nas będą interesować dwa:
- user → nazwa użytkownika bazy danych
- password → hasło wskazanego użytkownika.

Co zauważalne, zarówno pierwszy jak i drugi parametr przekazywany będzie otwartym tekstem. Warto w tym miejscu pomyśleć o funkcji „zaciemniającej" podawanie hasła, samo zaś hasło dobrze byłoby przekazywać jako zmienną pozyskaną z funkcji szyfrującej/deszyfrującej (chociaż osoba posiadająca nasz program i tak może wejść w posiadanie takiego hasła). W naszym wypadku użytkownik nie posiada hasła, stąd po znaku = nie ma żadnej wartości.

Jak można wywnioskować, połączenie do bazy w wykonaniu sterownika Java to nic innego jak adres URL do zasobu.

Jeżeli połączenie odniesie sukces, do zmiennej conn zwrócony zostanie adres-uchwyt połączenia do bazy danych (nie jest to zmienna z określoną wartością, a adres połączenia ustanowionego w systemie operacyjnym).

2) spróbujmy cokolwiek pobrać z bazy danych. W tym celu wykorzystamy ustanowione połączenie:

```
stmt = conn.createStatement();
```

w ten sposób przypiszemy nasze połączenie do klasy umożliwiającej operacje na obiektach bazy danych.

Z kolei w ten sposób:

```
rs = stmt.executeQuery("SELECT * FROM `miejscowosci`");
```

wykonamy poleczenie SQL wybierające WSZYSTKIE wiersze ze WSZYSTKIMI polami wskazanej tabeli (więcej o poleceniach http://planetatechnika.pl/SBD/mysql.pdf oraz http://planetatechnika.pl/SBD/mysql.pdf).
Należy pamiętać, że to jedynie przypisze nam połączenie uchwyt do wyników z tabeli. Aby przemieszczać się po kolejnych wynikach (rekordach z bazy danych) trzeba użyć odpowiedniej funkcji, jaką jest:

```
rs.next();
```

Metoda ta zwraca wartość true do chwili, gdy rs przechodzi na kolejny wierz z danymi. Należy przy tym pamiętać, że wstępnie rs nie ma załadowanego ŻADNEGO wiersza danych (stąd przed odczytem nawet pierwszego wiersza wymagane jest wykonanie tej funkcji).

3) Uniwersalnie, by odczytać wszystkie wyniki (wiersze) z naszego zapytania można zaimplementować taki kod:

```
 while(rs.next())
        System.out.println("Nazwa miasta " + rs.getString(2));
```

Powyższy kod będzie się wykonywać do chwili aż Java nie napotka braku kolejnych wierszy danych. Za każdym razem będziemy mieli wyświetloną nazwę miasta, która jest w drugiej kolumnie naszego wyniku (pierwszy to identyfikator wiersza).

Należy przy tym pamiętać,że ResultSet posiada kilka metod pobierania danych:

getInt(int id)
getFloat(int id)
getLong(int id)
getDouble(int id)

Najbardziej uniersalna jest oczywiście getString, ponieważ baza danych bez problemu potrafi przemienić dowolny przechowywany typ właśnie na ciąg znakowy. Ponadto wymienione metody posiadają swoje przeciążenie, w którym zamiast identyfikatora kolumny można podawać jej nazwę (np. „nazwa” zamiast 2).

UWAGA! Kolumny nie są liczone od 0 lecz od wartości 1!

4) Na koniec warto pamiętać o zamknięciu połączenia (zwolnienie ewentualnej blokady bazy danych, zmniejszenie ryzyka wycieku i niespójności danych itp.)

conn.close()

Cały kod może wyglądać następująco:

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
try {
	conn = DriverManager.getConnection("jdbc:mysql://localhost/biblioteka_java?" +
                                                    "user=root&password=");
	stmt = conn.createStatement();
	rs = stmt.executeQuery("SELECT * FROM miejscowosci");
	while(rs.next())
		System.out.println("Nazwa miejscowości " + rs.getString(2));
	conn.close();
}
catch (Exception ex) {
	System.out.println("Nie połączono! " + ex.getMessage());
}
```

5) Statement posiada prócz metody executeQuery także metodę execute. Nie zwraca ona wyników lecz wartość prawda/fałsz (wykonano/nie wykonano), zaś sama zmienna Statement przechowuje wynik polecenia (można go przypisać do ResultSet poprzez np. rs = stmt.getResultSet();) Metodę tę można jednak wykorzystać np. do wykonywania poleceń:

CREATE
ALTER
INSERT
UPDATE
DELETE

i każdego innego, które nie zwraca nam żadnych wyników do wyświetlenia.

Trzeba pamiętać, że wykonywanie poleceń jedno po drugim nie jest efektywną metodą pracy z bazą danych, szczególnie w przypadku, gdy musimy dodać/zamienić/usunąć jednocześnie wiele rekordów. Możemy wtedy zarządzać sesjami nie tylko z poziomu SQL, ale także sterownika.

Materiały:
https://www.javatpoint.com/example-to-connect-to-the-mysql-database
https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-installing-classpath.html
https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-usagenotes-connect-drivermanager.html
https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-url-format.html
https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-usagenotes-statements.html