Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Aplikacyjne (PAP)

JDBC

Julian Myrcha
Institute of Computer Science
26.02.2025

Historia (zatacza koło ...)

- najpierw każdy producent dostarczał biblioteki do języka C

Historia (zatacza koło ...)

- najpierw każdy producent dostarczał biblioteki do języka C
- w celu poprawienia czytelności wymyślono `embedded sql` - czyli preprocesor pozwalający na pisanie gołego SQL w kodzie

Historia (zatacza koło ...)

- najpierw każdy producent dostarczał biblioteki do języka C
- w celu poprawienia czytelności wymyślono `embedded sql` - czyli preprocesor pozwalający na pisanie gołego SQL w kodzie
- potem Microsoft zrobił `ODBC` (to jeszcze standard proceduralny)

Historia (zatacza koło ...)

- najpierw każdy producent dostarczał biblioteki do języka C
- w celu poprawienia czytelności wymyślono `embedded sql` - czyli preprocesor pozwalający na pisanie gołego SQL w kodzie
- potem Microsoft zrobił `ODBC` (to jeszcze standard proceduralny)
  - mimo 20 lat drivery ODBC są do każdej bazy danych na rynku

Historia (zatacza koło ...)

- najpierw każdy producent dostarczał biblioteki do języka C
- w celu poprawienia czytelności wymyślono `embedded sql` - czyli preprocesor pozwalający na pisanie gołego SQL w kodzie
- potem Microsoft zrobił `ODBC` (to jeszcze standard proceduralny)
  - mimo 20 lat drivery ODBC są do każdej bazy danych na rynku
  - driver implementuje część (daną przez poziom) zamkniętej specyfikacji

Historia (zatacza koło ...)

- najpierw każdy producent dostarczał biblioteki do języka C
- w celu poprawienia czytelności wymyślono `embedded sql` - czyli preprocesor pozwalający na pisanie gołego SQL w kodzie
- potem Microsoft zrobił `ODBC` (to jeszcze standard proceduralny)
  - mimo 20 lat drivery ODBC są do każdej bazy danych na rynku
  - driver implementuje część (daną przez poziom) zamkniętej specyfikacji
  - baza widziana po nazwie, co nazwa oznacza ustalane w administratorze ODBC na poziomie systemu

Historia (zatacza koło ...)

- najpierw każdy producent dostarczał biblioteki do języka C
- w celu poprawienia czytelności wymyślono `embedded sql` - czyli preprocesor pozwalający na pisanie gołego SQL w kodzie
- potem Microsoft zrobił `ODBC` (to jeszcze standard proceduralny)
  - mimo 20 lat drivery ODBC są do każdej bazy danych na rynku
  - driver implementuje część (daną przez poziom) zamkniętej specyfikacji
  - baza widziana po nazwie, co nazwa oznacza ustalane w administratorze ODBC na poziomie systemu
  - driver ODBC korzysta z bibliotek klienta danej bazy

Historia (zatacza koło ...)

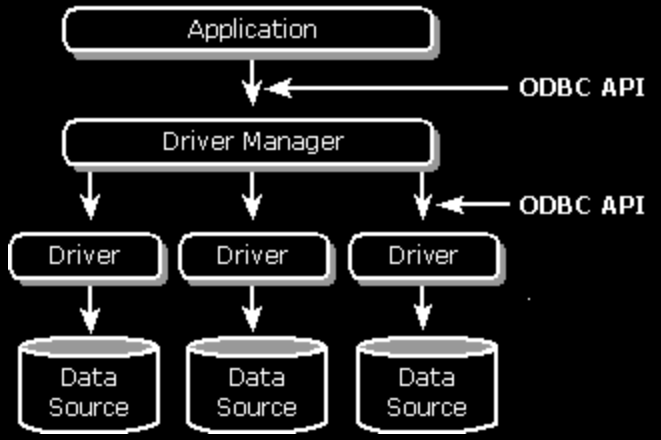
- najpierw każdy producent dostarczał biblioteki do języka C
- w celu poprawienia czytelności wymyślono `embedded sql` - czyli preprocesor pozwalający na pisanie gołego SQL w kodzie
- potem Microsoft zrobił `ODBC` (to jeszcze standard proceduralny)
  - mimo 20 lat drivery ODBC są do każdej bazy danych na rynku
  - driver implementuje część (daną przez poziom) zamkniętej specyfikacji
  - baza widziana po nazwie, co nazwa oznacza ustalane w administratorze ODBC na poziomie systemu
  - driver ODBC korzysta z bibliotek klienta danej bazy
- potem Microsoft zrobił `ADO` (czyli obiektowy interfejs w technologii COM)

Historia (zatacza koło ...)

- najpierw każdy producent dostarczał biblioteki do języka C
- w celu poprawienia czytelności wymyślono `embedded sql` - czyli preprocesor pozwalający na pisanie gołego SQL w kodzie
- potem Microsoft zrobił `ODBC` (to jeszcze standard proceduralny)
  - mimo 20 lat drivery ODBC są do każdej bazy danych na rynku
  - driver implementuje część (daną przez poziom) zamkniętej specyfikacji
  - baza widziana po nazwie, co nazwa oznacza ustalane w administratorze ODBC na poziomie systemu
  - driver ODBC korzysta z bibliotek klienta danej bazy
- potem Microsoft zrobił `ADO` (czyli obiektowy interfejs w technologii COM)
- a potem `ADO.NET`

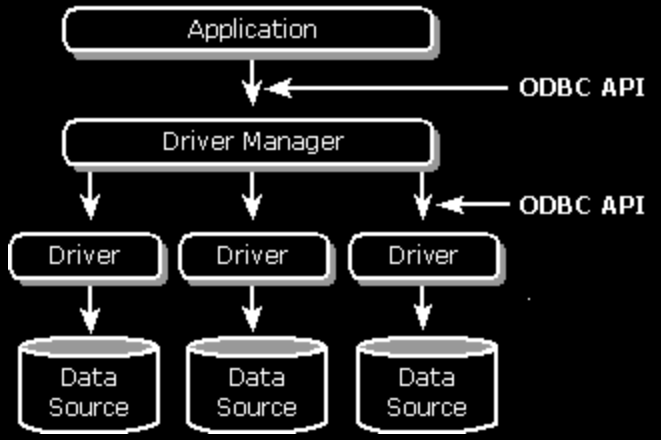
Architektura ODBC składa się z czterech komponentów:

- `Application` Przetwarza dane pobrane za pomocą ODBC. Woła ODBC w celu wykonania funkcji związanych z definicją, pobraniem i zapisem danych



Architektura ODBC składa się z czterech komponentów:

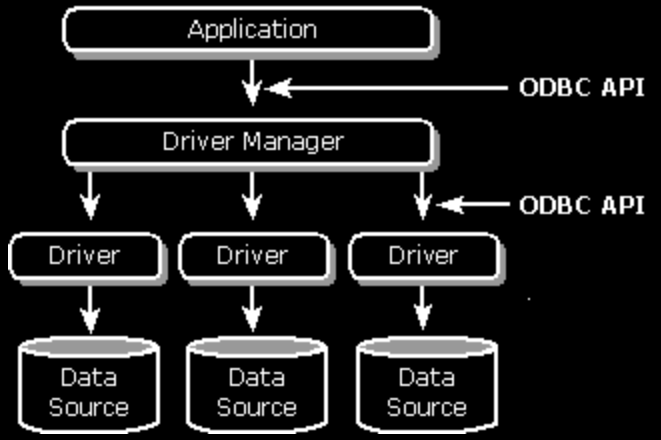
- `Application` Przetwarza dane pobrane za pomocą ODBC. Woła ODBC w celu wykonania funkcji związanych z definicją, pobraniem i zapisem danych



- `Driver Manager` Ładuje i zwalnia drivery potrzebne przez aplikację. Przetwarza żądania aplikacji i przekazuje je do odpowiedniego drivera.

Architektura ODBC składa się z czterech komponentów:

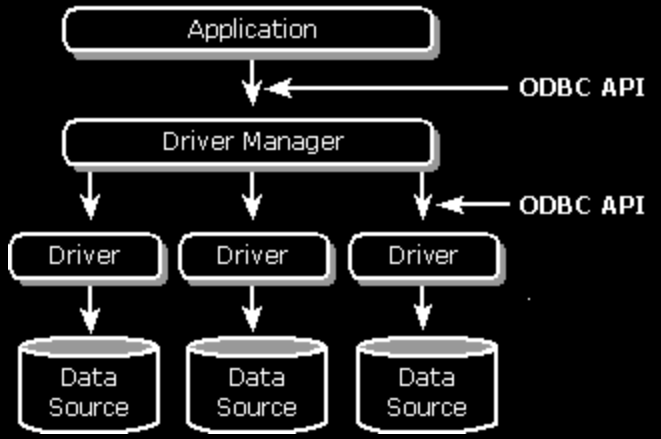
- `Application` Przetwarza dane pobrane za pomocą ODBC. Woła ODBC w celu wykonania funkcji związanych z definicją, pobraniem i zapisem danych



- `Driver Manager` Ładuje i zwalnia drivery potrzebne przez aplikację. Przetwarza żądania aplikacji i przekazuje je do odpowiedniego drivera.
- `Driver` przetwarza wywołania ODBC, przekazuje polecania SQL, w razie konieczności modyfikuje je tak, aby były zgodne z typem bazy.

Architektura ODBC składa się z czterech komponentów:

- `Application` Przetwarza dane pobrane za pomocą ODBC. Woła ODBC w celu wykonania funkcji związanych z definicją, pobraniem i zapisem danych



- `Driver Manager` Ładuje i zwalnia drivery potrzebne przez aplikację. Przetwarza żądania aplikacji i przekazuje je do odpowiedniego drivera.
- `Driver` przetwarza wywołania ODBC, przekazuje polecania SQL, w razie konieczności modyfikuje je tak, aby były zgodne z typem bazy.
- `Data source` Dane (baza) z którą się komunikujemy.

- SQL to skrót od Structured Query Language

- SQL to skrót od Structured Query Language
- SQL umożliwia dostęp do baz danych - do danych i zarządzania

- SQL to skrót od Structured Query Language
- SQL umożliwia dostęp do baz danych - do danych i zarządzania
- SQL stał się standardem:
  - American National Standards Institute (ANSI) w 1986 r.

- SQL to skrót od Structured Query Language
- SQL umożliwia dostęp do baz danych - do danych i zarządzania
- SQL stał się standardem:
  - American National Standards Institute (ANSI) w 1986 r.
  - Międzynarodowej Organizacji Normalizacyjnej (ISO) w 1987 r

- SQL to skrót od Structured Query Language
- SQL umożliwia dostęp do baz danych - do danych i zarządzania
- SQL stał się standardem:
  - American National Standards Institute (ANSI) w 1986 r.
  - Międzynarodowej Organizacji Normalizacyjnej (ISO) w 1987 r

bardzo dobre wprowadzenie do SQL jest na stronie `https://www.w3schools.com/sql/` (trzeba mieć chrome aby dobrze działały przykłady)

- SQL to skrót od Structured Query Language
- SQL umożliwia dostęp do baz danych - do danych i zarządzania
- SQL stał się standardem:
  - American National Standards Institute (ANSI) w 1986 r.
  - Międzynarodowej Organizacji Normalizacyjnej (ISO) w 1987 r
- SQL to język deklaratywny - nie mówi jak ale co chcemy uzyskać!

- utworzenie tablicy

```sql
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255)
);
```

- utworzenie tablicy
- utworzenie tablicy z kluczem

```sql
CREATE TABLE Persons (
    PersonID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    CONSTRAINT PK_Person PRIMARY KEY (PersonID)
);
```

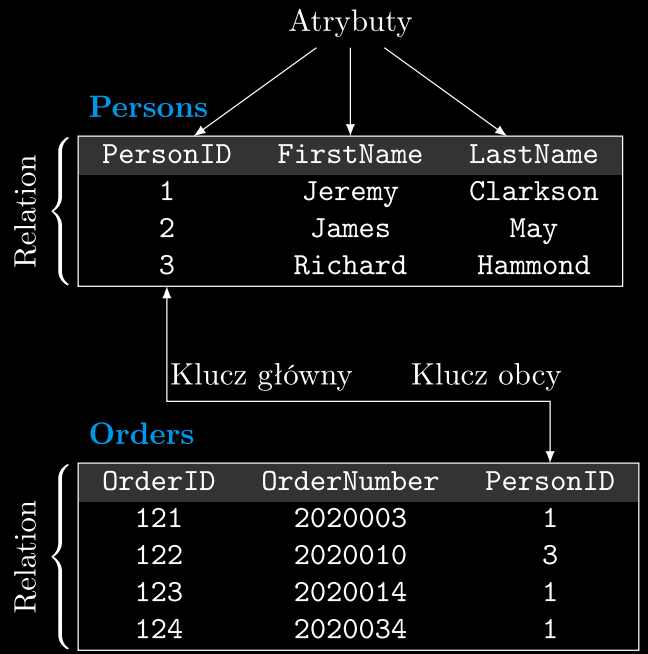
- utworzenie tablicy
- utworzenie tablicy z kluczem
  - klucz może być generowany przez bazę (tu składnia Derby-JavaDB)

```sql
CREATE TABLE Persons (
    PersonID int NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    CONSTRAINT PK_Person PRIMARY KEY (PersonID)
);
```

- utworzenie tablicy
- utworzenie tablicy z kluczem
  - klucz może być generowany przez bazę (tu składnia Derby-JavaDB)
- utworzenie tablicy z kluczem obcym

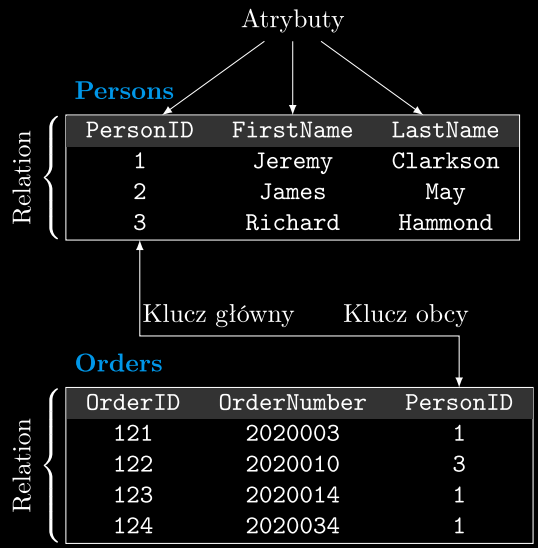
```
CREATE TABLE Persons (
    PersonID int NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    CONSTRAINT PK_Person PRIMARY KEY (PersonID)
);
CREATE TABLE Orders (
    OrderID int NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
                              REFERENCES Persons(PersonID)
);
```

- utworzenie tablicy
- utworzenie tablicy z kluczem
  - klucz może być generowany przez bazę (tu składnia Derby-JavaDB)
- utworzenie tablicy z kluczem obcym
  - baza pilnuje, aby istniał rekord do którego się odwołujemy



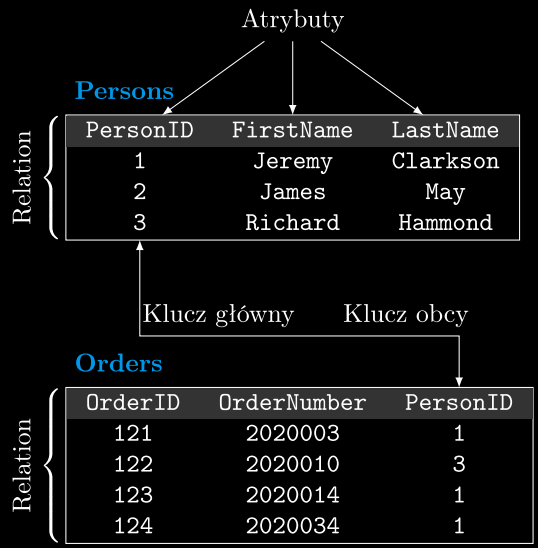
```sql
INSERT INTO Persons (FirstName, LastName) VALUES ('Jeremy', 'Clarkson');
INSERT INTO Persons (FirstName, LastName) VALUES ('James', 'May');
INSERT INTO Persons (FirstName, LastName) VALUES ('Richard', 'Hammond');
GO;
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020003, 1);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020010, 3);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020014, 1);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020034, 1);
GO;
```

- w zależności od bazy klucz może być:



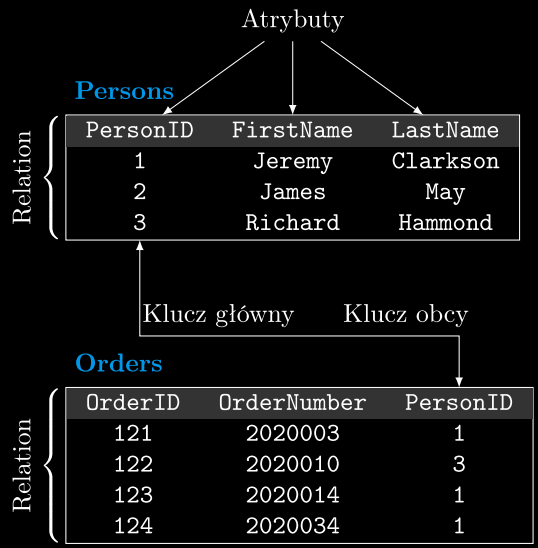
```sql
INSERT INTO Persons (FirstName, LastName) VALUES ('Jeremy', 'Clarkson');
INSERT INTO Persons (FirstName, LastName) VALUES ('James', 'May');
INSERT INTO Persons (FirstName, LastName) VALUES ('Richard', 'Hammond');
GO;
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020003, 1);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020010, 3);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020014, 1);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020034, 1);
GO;
```

- w zależności od bazy klucz może być:
  - ustawiany przez programistę



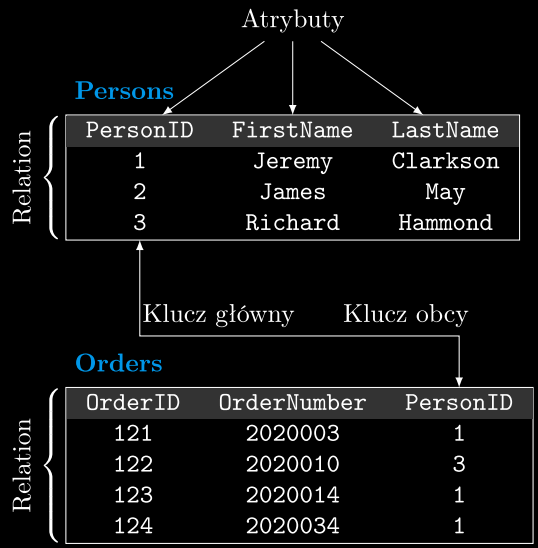
```sql
INSERT INTO Persons (FirstName, LastName) VALUES ('Jeremy', 'Clarkson');
INSERT INTO Persons (FirstName, LastName) VALUES ('James', 'May');
INSERT INTO Persons (FirstName, LastName) VALUES ('Richard', 'Hammond');
GO;
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020003, 1);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020010, 3);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020014, 1);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020034, 1);
GO;
```

- w zależności od bazy klucz może być:
  - ustawiany przez programistę
  - ustawiany przez bazę



```sql
INSERT INTO Persons (FirstName, LastName) VALUES ('Jeremy', 'Clarkson');
INSERT INTO Persons (FirstName, LastName) VALUES ('James', 'May');
INSERT INTO Persons (FirstName, LastName) VALUES ('Richard', 'Hammond');
GO;
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020003, 1);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020010, 3);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020014, 1);
INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020034, 1);
GO;
```

- w zależności od bazy klucz może być:
  - ustawiany przez programistę
  - ustawiany przez bazę
- klucz obcy zawsze ustawia programista:



- z jednej tablicy

```sql
SELECT * FROM Persons WHERE FirstName LIKE '%J%';
```

Persons

Relation

PersonID	FirstName	LastName
1	Jeremy	Clarkson
2	James	May
3	Richard	Hammond

SELECT

Relation

PersonID	FirstName	LastName
1	Jeremy	Clarkson
2	James	May

- z jednej tablicy
- z wielu tabel

```sql
SELECT Orders.OrderNumber, Persons.FirstName, Persons.LastName
FROM Orders
INNER JOIN Persons ON Orders.PersonID = Persons.PersonID;
```

Persons

Relation

PersonID	FirstName	LastName
1	Jeremy	Clarkson
2	James	May
3	Richard	Hammond

Orders

Relation

OrderID	OrderNumber	PersonID
121	2020003	1
122	2020010	3
123	2020014	1
124	2020034	1

SELECT

Relation

OrderNumber	FirstName	LastName
2020003	Jeremy	Clarkson
2020010	Richard	Hammond
2020014	Jeremy	Clarkson
2020034	Jeremy	Clarkson

MODYFIKACJA DANYCH (UPDATE)

```sql
UPDATE Orders
    SET OrderNumber = 20200015, PersonID= 3
    WHERE OrderID = 123;
```

USUNIĘCIE DANYCH (DELETE)

```sql
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

USUNIĘCIE DANYCH (DELETE)

```sql
DELETE FROM Orders WHERE OrderNumber=2020034;
```

- JDBC to skrót od Java Database Connectivity

CO TO JEST JDBC?

- JDBC to skrót od Java Database Connectivity
- Standardowy interfejs API języka Java zapewniający niezależną od bazy danych łączność między językiem programowania Java a szeroką gamą baz danych.

- JDBC to skrót od Java Database Connectivity
- Standardowy interfejs API języka Java zapewniający niezależną od bazy danych łączność między językiem programowania Java a szeroką gamą baz danych.
- Biblioteka JDBC zawiera interfejsy API dla każdego z wymienionych poniżej zadań, które są często związane z użyciem bazy danych
  - Nawiązywanie połączenia z bazą danych

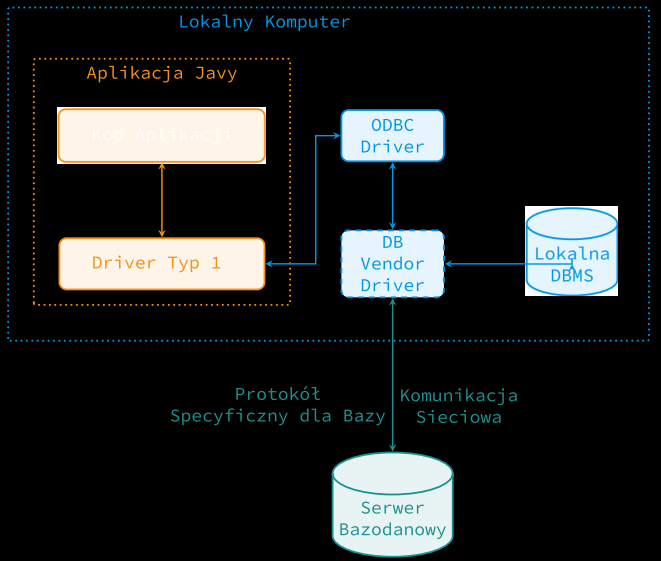
- JDBC to skrót od Java Database Connectivity
- Standardowy interfejs API języka Java zapewniający niezależną od bazy danych łączność między językiem programowania Java a szeroką gamą baz danych.
- Biblioteka JDBC zawiera interfejsy API dla każdego z wymienionych poniżej zadań, które są często związane z użyciem bazy danych
  - Nawiązywanie połączenia z bazą danych
  - Tworzenie instrukcji SQL

- JDBC to skrót od Java Database Connectivity
- Standardowy interfejs API języka Java zapewniający niezależną od bazy danych łączność między językiem programowania Java a szeroką gamą baz danych.
- Biblioteka JDBC zawiera interfejsy API dla każdego z wymienionych poniżej zadań, które są często związane z użyciem bazy danych
  - Nawiązywanie połączenia z bazą danych
  - Tworzenie instrukcji SQL
  - Wykonywanie zapytań SQL w bazie danych

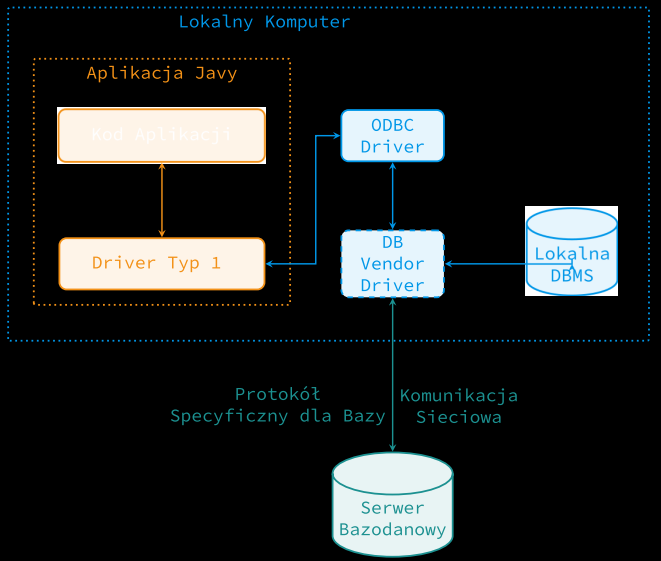
- JDBC to skrót od Java Database Connectivity
- Standardowy interfejs API języka Java zapewniający niezależną od bazy danych łączność między językiem programowania Java a szeroką gamą baz danych.
- Biblioteka JDBC zawiera interfejsy API dla każdego z wymienionych poniżej zadań, które są często związane z użyciem bazy danych
  - Nawiązywanie połączenia z bazą danych
  - Tworzenie instrukcji SQL
  - Wykonywanie zapytań SQL w bazie danych
  - Przeglądanie i modyfikowanie wynikowych rekordów

- JDBC to skrót od Java Database Connectivity
- Standardowy interfejs API języka Java zapewniający niezależną od bazy danych łączność między językiem programowania Java a szeroką gamą baz danych.
- Biblioteka JDBC zawiera interfejsy API dla każdego z wymienionych poniżej zadań, które są często związane z użyciem bazy danych
  - Nawiązywanie połączenia z bazą danych
  - Tworzenie instrukcji SQL
  - Wykonywanie zapytań SQL w bazie danych
  - Przeglądanie i modyfikowanie wynikowych rekordów
- Zasadniczo JDBC to specyfikacja zapewniająca pełny zestaw interfejsów, który umożliwia przenośny dostęp do podstawowej bazy danych.

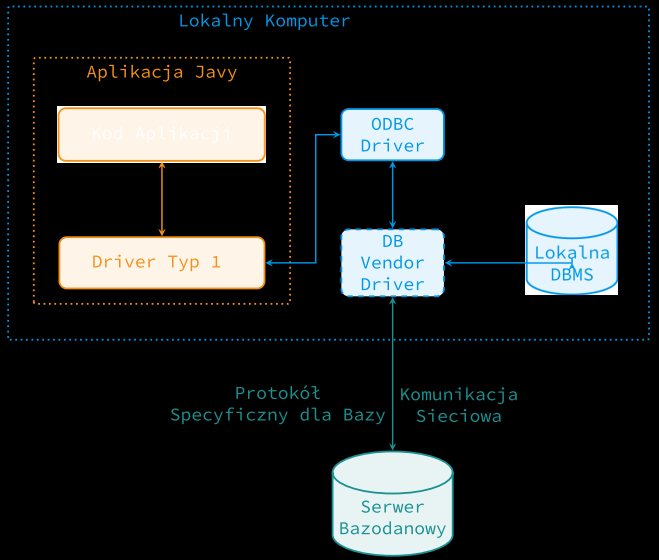
- wywołuje natywny kod lokalnie dostępnego sterownika ODBC



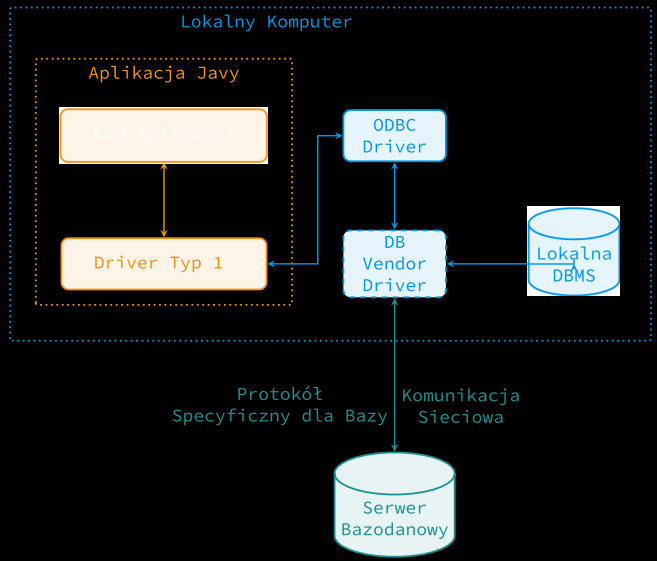
- wywołuje natywny kod lokalnie dostępnego sterownika ODBC
- Driver ODBC musi być na maszynie klienta



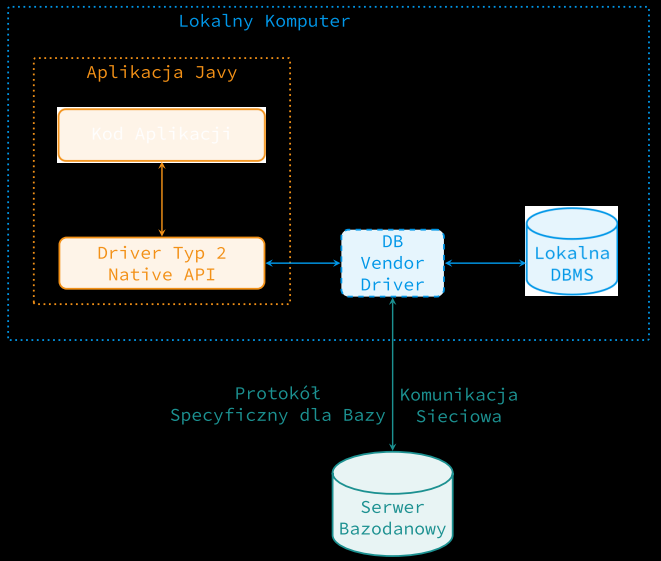
- wywołuje natywny kod lokalnie dostępnego sterownika ODBC
- Driver ODBC musi być na maszynie klienta
- W JDBC 4.2 most JDBC-ODBC został usunięty



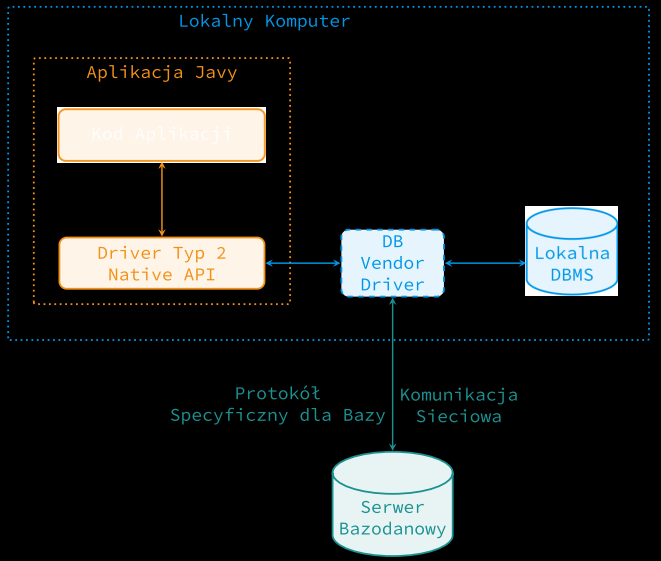
- wywołuje natywny kod lokalnie dostępnego sterownika ODBC
- Driver ODBC musi być na maszynie klienta
- W JDBC 4.2 most JDBC-ODBC został usunięty
- Brak wsparcia od wersji Java 8



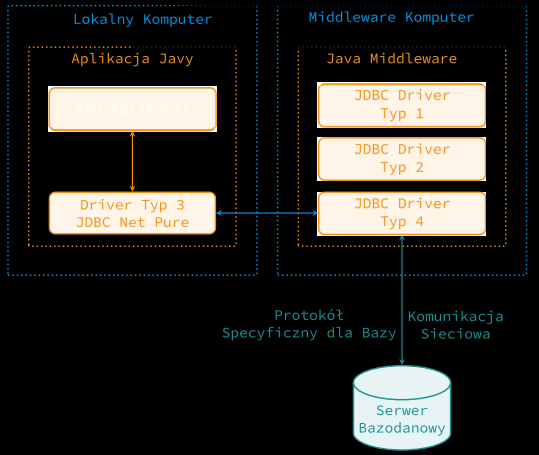
- wywołuje natywną bibliotekę dostawcy bazy danych po stronie klienta.



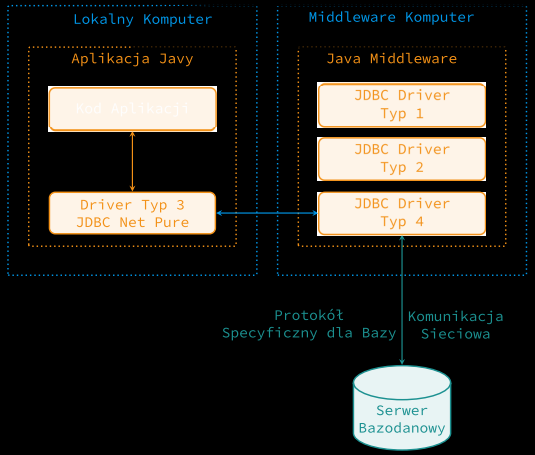
- wywołuje natywną bibliotekę dostawcy bazy danych po stronie klienta.
- natywna biblioteka następnie komunikuje się z bazą danych przez sieć.



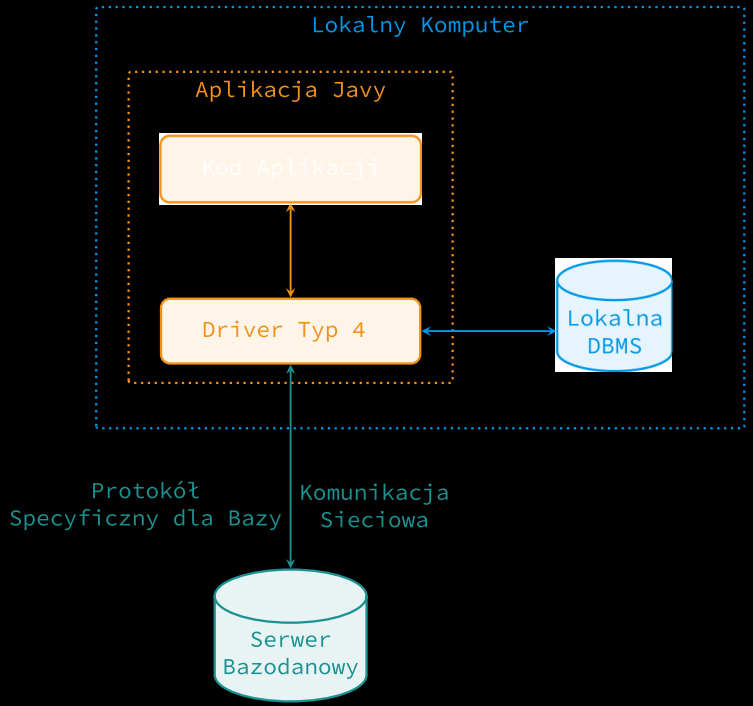
- sterownik w czystej Javie, który komunikuje się z oprogramowaniem pośrednim po stronie serwera



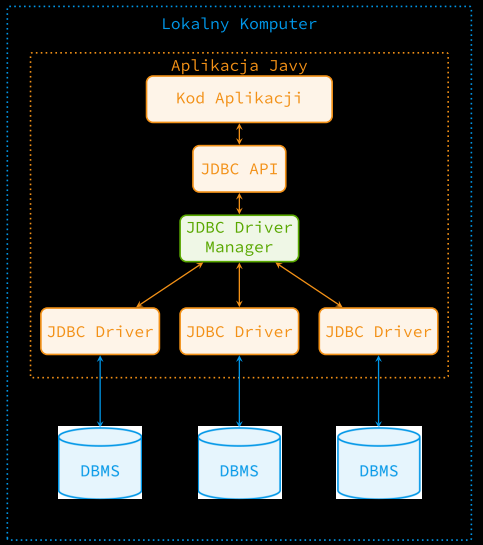
- sterownik w czystej Javie, który komunikuje się z oprogramowaniem pośrednim po stronie serwera
- oprogramowanie pośrednie następnie komunikuje się z bazą danych.



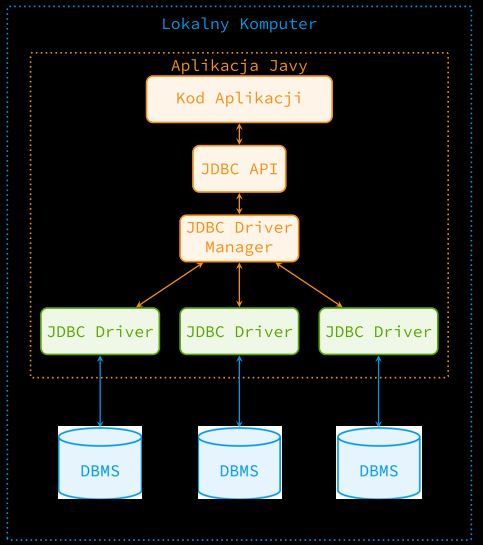
- sterownik w czystej Javie, który używa natywnego protokołu bazy danych.



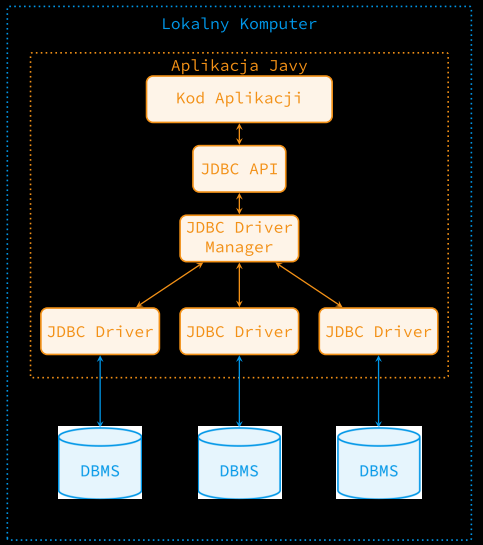
- DriverManager-Ta klasa zarządza listą sterowników bazy danych.
  - Dopasowuje żądania połączeń z aplikacji Java z odpowiednim sterownikiem bazy danych przy użyciu protokołu komunikacyjnego.
  - Do ustanowienia połączenia z bazą danych zostanie użyty pierwszy sterownik, który rozpozna określony podprotokół w JDBC.



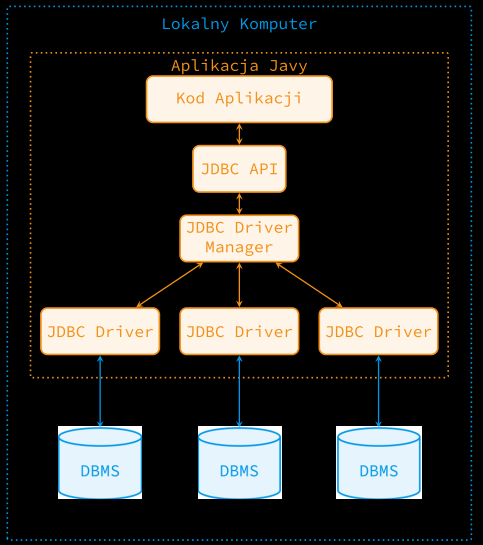
- DriverManager-Ta klasa zarządza listą sterowników bazy danych.
- Driver- Ten interfejs obsługuje komunikację z serwerem bazy danych
  - Potrzeba bezpośredniego odwoływania się do drivera występuje sporadycznie
  - Zamiast tego wykorzystujemy DriverManagera
  - DriverManager ukrywa szczegóły tej komunikacji



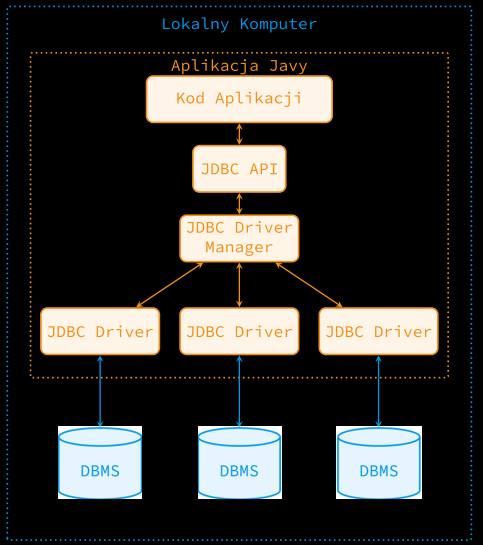
- DriverManager-Ta klasa zarządza listą sterowników bazy danych.
- Driver- Ten interfejs obsługuje komunikację z serwerem bazy danych
- Connection- interfejs ze wszystkimi metodami kontaktowania się z bazą danych.
  - Obiekt połączenia reprezentuje kontekst komunikacyjny
  - cała komunikacja z bazą danych odbywa się tylko przez obiekt połączenia.



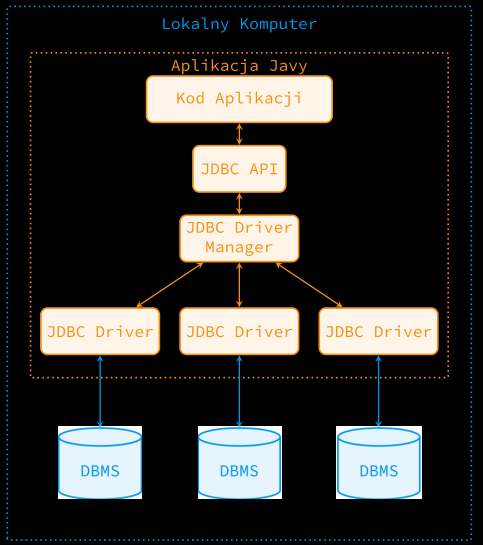
- DriverManager-Ta klasa zarządza listą sterowników bazy danych.
- Driver- Ten interfejs obsługuje komunikację z serwerem bazy danych
- Connection- interfejs ze wszystkimi metodami kontaktowania się z bazą danych.
- Statement- Obiekty utworzone za pomocą tego interfejsu służą do przesyłania instrukcji SQL do bazy danych.
  - Niektóre interfejsy pochodne akceptują parametry oprócz wykonywania procedur składowanych.



- DriverManager-Ta klasa zarządza listą sterowników bazy danych.
- Driver- Ten interfejs obsługuje komunikację z serwerem bazy danych
- Connection- interfejs ze wszystkimi metodami kontaktowania się z bazą danych.
- Statement- Obiekty utworzone za pomocą tego interfejsu służą do przesyłania instrukcji SQL do bazy danych.
- ResultSet- Te obiekty przechowują dane pobrane z bazy danych
  - po wykonaniu zapytania SQL przy użyciu Statement
  - Działa jako iterator, umożliwiający poruszanie się po jego danych.



- DriverManager-Ta klasa zarządza listą sterowników bazy danych.
- Driver- Ten interfejs obsługuje komunikację z serwerem bazy danych
- Connection- interfejs ze wszystkimi metodami kontaktowania się z bazą danych.
- Statement- Obiekty utworzone za pomocą tego interfejsu służą do przesyłania instrukcji SQL do bazy danych.
- ResultSet- Te obiekty przechowują dane pobrane z bazy danych
- SQLException- obsługuje wszelkie błędy, które występują w aplikacji bazodanowej.



- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`

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

Connection conn = null;
try {
  log.info("Opening connection to bookStoreDB");
  conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
} catch (SQLException ex) {
  log.error("Unable to open connection", ex);
}
```

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy

```
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.SQLException;
4
5  Connection conn = null;
6  try {
7    log.info("Opening connection to bookStoreDB");
8    conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
9  } catch (SQLException ex) {
10   log.error("Unable to open connection", ex);
11 }
```

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy
  - w przypadku błędu połączenia rzucany jest wyjątek `SQLException`

```
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.SQLException;
4
5  Connection conn = null;
6  try {
7    log.info("Opening connection to bookStoreDB");
8    conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
9  } catch (SQLException ex) {
10   log.error("Unable to open connection", ex);
11 }
```

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy
  - w przypadku błędu połączenia rzucany jest wyjątek `SQLException`
- zamknięcie połączenia wymaga wywołania metody `close()`

```
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.SQLException;
4
5  Connection conn = ...
6  try {
7    log.info("Closing database connection to bookStoreDB");
8    conn.close();
9  } catch (SQLException ex) {
10   log.error("Unable to close connection", ex);
11 }
```

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy
  - w przypadku błędu połączenia rzucany jest wyjątek `SQLException`
- zamknięcie połączenia wymaga wywołania metody `close()`
  - trzeba zrobić ręcznie, bo sprzątaczka zawoła gdy będzie zwalniała pamięć co może się opóźnić

```
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.SQLException;
4
5  Connection conn = ...
6  try {
7    log.info("Closing database connection to bookStoreDB");
8    conn.close();
9  } catch (SQLException ex) {
10   log.error("Unable to close connection", ex);
11 }
```

```java
package pap;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBContext implements AutoCloseable {
  private static final Logger log = LoggerFactory.getLogger(DBContext.class);
  private Connection conn = null;

  public void close() {
    if (conn != null) {
      try {
        log.info("Closing database connection to bookStoreDB");
        conn.close();
      } catch (SQLException ex) {
        log.error("Unable to close connection", ex);
      }
      conn = null;
    }
  }

  public Connection getConnection() throws SQLException { // zasob nie w konstruktorze!
    if (conn == null) {
      log.info("Opening connection to bookStoreDB");
      conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
    }
    return conn;
  }
}
```

- obiekt `Connection` uzyskujemy za pomocą wywołania statycznej metody `getConnection` klasy `DriverManager`
  - na podstawie connection stringa wybierana jest baza danych do której się łączymy
  - w przypadku błędu połączenia rzucany jest wyjątek `SQLException`
- zamknięcie połączenia wymaga wywołania metody `close()`
  - trzeba zrobić ręcznie, bo sprzątaczka zawoła gdy będzie zwalniała pamięć co może się opóźnić
  - można wykorzystać AutoCloseable

```
1 package pap;
2
3 try(var ctx = new DBContext()) {
4     var conn = ctx.getConnection();  // klasa Context implementuje AutoCloseable
5     ...
6 }
```

```
Connection conn = ... ;
Statement stmt = null;
try {
    stmt = conn.createStatement();
    stmt.execute(
        "CREATE TABLE bookstore ("+
        "id INT IDENTITY,"+
        " ISBN VARCHAR(30),"+
        " title VARCHAR(30),"+
        " pages INT)");
    log.info("Creating table");
    success = true;
} catch (SQLException e) {
    log.error("Unable to create the database table", e);
} finally {
    if (stmt != null)
        try {
            stmt.close();
        } catch (SQLException e) {
        }
}
```

derby - ”jdbc:derby:./data;create=true” - baza w pliku, w katalogu data

pom.xml

```
    <!-- JDBC Connector for DERBY -->
    <dependency>
      <groupId>org.apache.derby</groupId>
      <artifactId>derby</artifactId>
      <version>10.15.2.0</version>
    </dependency>
```

derby - ”jdbc:derby:./data;create=true” - baza w pliku, w katalogu data

sqlserver - ”jdbc:sqlserver://172.17.0.2;databaseName=TestDB;”

pom.xml

```
<!-- https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc -->
<dependency>
  <groupId>com.microsoft.sqlserver</groupId>
  <artifactId>mssql-jdbc</artifactId>
  <version>9.1.1.jre15-preview</version>
</dependency>
```

derby - ”jdbc:derby:./data;create=true” - baza w pliku, w katalogu data

sqlserver - ”jdbc:sqlserver://172.17.0.2;databaseName=TestDB;”

mysql -”jdbc:mysql://localhost:3306/pap”

pom.xml

```
<!-- JDBC Connector for MySQL -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.16</version>
</dependency>
```

derby - ”jdbc:derby:./data;create=true” - baza w pliku, w katalogu data

sqlserver - ”jdbc:sqlserver://172.17.0.2;databaseName=TestDB;”

mysql -”jdbc:mysql://localhost:3306/pap”

oracle -”jdbc:oracle:thin:@localhost:51521/XEPDB1”

pom.xml

```
    <dependency>
      <groupId>com.oracle.database.jdbc</groupId>
      <artifactId>ojdbc8-production</artifactId>
      <version>19.7.0.0</version>
      <type>pom</type>
    </dependency>
```

- synctactic sugar

```
class Foo implements AutoCloseable {
  public void method() {
    System.out.println("method");
  }
  public void close() {
    System.out.println("close");
  }
}
public class App {
  public static void main(String[] args) throws Exception {
    System.out.println("before");
    try (Foo f = new Foo()) {
      f.method();
    }



    System.out.println("after");
  }
}
```

- synctactic sugar
- jest równoważne

```java
class Foo implements AutoCloseable {
  public void method() {
    System.out.println("method");
  }
  public void close() {
    System.out.println("close");
  }
}
public class App {
  public static void main(String[] args) throws Exception {
    System.out.println("before");
    Foo f = new Foo();
    try  {
      f.method();
    } finally {
      f.close();
    }
    System.out.println("after");
  }
}
```

- synctactic sugar
- jest równoważne
- oba drukują:

```
before
method
close
after
```

- synctactic sugar
- jest równoważne
- oba drukują:
- działa gdy przerwanie po alokacji obiektu

```
class Foo implements AutoCloseable {
  public void method() {
    System.out.println("method");
  }
  public void close() {
    System.out.println("close");
  }
}
public class App {
  public static void main(String[] args) throws Exception {
    System.out.println("before");
    Foo f = new Foo();
    try  {
      f.method();
    } finally {
      f.close();
    }
    System.out.println("after");
  }
}
```

- klasa jest długa

```
public class Book {
  private int id;
  private String ISBN;
  private String title;
  private int pages;

  public int getId() {
    return id;
  }
...
```

- klasa jest długa
- narzędzia takie jak lombok ułatwiają pisanie tego typu nudnego kodu

```
public class Book {
  private int id;
  private String ISBN;
  private String title;
  private int pages;

  public int getId() {
    return id;
  }
...
```

```java
public class DBContext {
  private static final Logger log = LoggerFactory.getLogger(DBContext.class);
  private Connection conn = null;
  public void close() {
    if (conn != null) {
      try {
        log.info("Closing database connection to bookStoreDB");
        conn.close();
      } catch (SQLException ex) {
        log.error("Unable to close connection", ex);
      }
      conn = null;
    }
  }
  public Connection getConnection() throws SQLException {
    if (conn == null) {
      log.info("Opening connection to bookStoreDB");
      conn = DriverManager.getConnection("jdbc:hsqldb:mem:bookStoreDB", "SA", "");
    }
    return conn;
  }
}
```

```
public class Main {
  static private BookStoreDAO dao = null;
  static private DBContext dbcon = null;

  public static void main(String[] argv) throws SQLException {
        dbcon = new DBContext();
        dao = new BookStoreDAO();
        dao.setConn(dbcon.getConnection());
        dao.createTable();

        Book book = new Book();
        book.setISBN("978-0544003415");
        book.setTitle("The Lord of the Rings");
        book.setPages(1216);
        dao.insertBook(book);

        List<Book> books = dao.getAllBooks();
        ...
```

- hsqldb

```
public boolean createTable() {
  boolean success = false;
  if (conn != null) {
    Statement stmt = null;
    try {
      stmt = conn.createStatement();
      stmt.execute("CREATE TABLE bookstore ("+
                   "id INT IDENTITY, ISBN VARCHAR(30),"+
                   "title VARCHAR(30), pages INT)");
      log.info("Creating table");
      success = true;
    } catch (SQLException e) {
      log.error("Unable to create the database table", e);
    } finally {
      if (stmt != null) {
        try {
          stmt.close();
        } catch (SQLException e) {
        }
      }
    }
  }
  return success;
}
```

- derby

```
public boolean createTable() {
  boolean success = false;
  if (conn != null) {
    Statement stmt = null;
    try {
      stmt = conn.createStatement();
      stmt.execute("CREATE TABLE bookstore ("+
           " id INT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1), "+
           " ISBN VARCHAR(30), title VARCHAR(30), pages INT)");
      log.info("Creating table");
      success = true;
    } catch (SQLException e) {
      log.error("Unable to create the database table", e);
    } finally {
      if (stmt != null) {
        try {
          stmt.close();
        } catch (SQLException e) {
        }
      }
    }
  }
  return success;
}
```

```
public boolean dropTable() {
  boolean success = false;
  if (conn != null) {
    Statement stmt = null;
    try {
      stmt = conn.createStatement();
      stmt.execute("DROP TABLE bookstore");
      log.info("Deleting table");
      success = true;
    } catch (SQLException e) {
      log.error("Unable to create the database table", e);
    } finally {
      if (stmt != null) {
        try {
          stmt.close();
        } catch (SQLException e) {
          log.error("close error", e);
        }
      }
    }
  }
  return success;
}
```

- hsqldb

```
1  public boolean insertBook(Book book) {
2    boolean success = false;
3    if (book != null) {
4      log.debug("insertBook({} {})", book.getISBN(), book.getTitle());
5      if (conn != null) {
6        PreparedStatement pstmt = null;
7        try {
8          pstmt = conn.prepareStatement("INSERT INTO bookstore VALUES (null, ?, ?, ?)");
9          pstmt.setString(1, book.getISBN());
10         pstmt.setString(2, book.getTitle());
11         pstmt.setInt(3, book.getPages());
12         success = (pstmt.executeUpdate() == 1); // sukces oznacza wstawienie dokladnie jednego rekordu
13       } catch (SQLException e) {
14         log.error("Unable to insert" + book.getISBN() + " " + book.getTitle() + "into the table", e);
15       } finally {
16         if (pstmt != null) {
17           try {
18             pstmt.close();
19           } catch (SQLException e) {
20             log.error("close error", e);
21           }
22         }
23       }
24     }
25   }
26   return success;
27 }
```

- derby

```
1  public boolean insertBook(Book book) {
2    boolean success = false;
3    if (book != null) {
4      log.debug("insertBook({} {})", book.getISBN(), book.getTitle());
5      if (conn != null) {
6        PreparedStatement pstmt = null;
7        try {
8          pstmt = conn.prepareStatement("INSERT INTO  bookstore (ISBN, Title,Pages) VALUES (?, ?, ?)");
9          pstmt.setString(1, book.getISBN());
10         pstmt.setString(2, book.getTitle());
11         pstmt.setInt(3, book.getPages());
12         success = (pstmt.executeUpdate() == 1); // sukces oznacza wstawienie dokladnie jednego rekordu
13       } catch (SQLException e) {
14         log.error("Unable to insert" + book.getISBN() + " " + book.getTitle() + "into the table", e);
15       } finally {
16         if (pstmt != null) {
17           try {
18             pstmt.close();
19           } catch (SQLException e) {
20             log.error("close error", e);
21           }
22         }
23       }
24     }
25   }
26   return success;
27 }
```

```java
public List<Book> getAllBooks() {
  List<Book> books = new ArrayList<Book>();
  if (conn != null) {
    Statement stmt = null;
    try {
      stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM bookstore");
      while (rs.next()) {
        Book book = new Book();
        book.setId(rs.getInt("id"));
        book.setISBN(rs.getString("ISBN"));
        book.setTitle(rs.getString("title"));
        book.setPages(rs.getInt("pages"));
        books.add(book);
      }
    } catch (SQLException e) {
      log.error("Unable to create the database table", e);
    } finally {
      if (stmt != null) {
        try { stmt.close();
        } catch (SQLException e) {
        }
      }
    }
  }
  return books;
}
```

getBookById(int id)

```
public Book getBookById(int id) { log.debug("getBookById({})", id);
  if (conn != null) {
    PreparedStatement pstmt = null;
    try {
      pstmt = conn.prepareStatement("SELECT * FROM bookstore WHERE id = ?");
      pstmt.setInt(1, id);
      ResultSet rs = pstmt.executeQuery();
      if (rs.next()) {
        Book book = new Book();
        book.setId(rs.getInt("id"));
        book.setISBN(rs.getString("ISBN"));
        book.setTitle(rs.getString("title"));
        book.setPages(rs.getInt("pages"));
      } else
        log.warn("No book found for id = {}", id);
    } catch (SQLException e) {
      log.error("Unable query book by ID(" + id + ")", e);
    } finally {
      if (pstmt != null) {
        try { pstmt.close();
        } catch (SQLException e) {
        }
      }
    }
  }
  return book;
}
```

Java DB (Oracle) i Derby (Apache) to to samo

- pobieramy derby ze strony

```
https://db.apache.org/derby/releases/release-10_15_2_0.cgi
```

Java DB (Oracle) i Derby (Apache) to to samo

- pobieramy derby ze strony
- rozpakowujemy do katalogu `/opt`

```
sudo mv db-derby-10.15.2.0-bin derby
sudo mv derby /opt
```

Java DB (Oracle) i Derby (Apache) to to samo

- pobieramy derby ze strony
- rozpakowujemy do katalogu `/opt`
- ustalenie zmiennej DERBY_HOME w pliku .bashrc

```
1 export DERBY_HOME=/opt/derby
```

Java DB (Oracle) i Derby (Apache) to to samo

- pobieramy derby ze strony
- rozpakowujemy do katalogu `/opt`
- ustalenie zmiennej DERBY_HOME w pliku .bashrc
- ustalenie ścieżki w pliku .bashrc

```
1 export PATH="$DERBY_HOME/bin:$PATH"
```

Java DB (Oracle) i Derby (Apache) to to samo

- pobieramy derby ze strony
- rozpakowujemy do katalogu `/opt`
- ustalenie zmiennej DERBY_HOME w pliku .bashrc
- ustalenie ścieżki w pliku .bashrc
- sprawdzenie poprawności instalacji - uruchomienie komendy ij

- po rozpakowaniu paczki `db-derby-10.15.2.0-bin.zip` do katalogu: `/opt/derby/`

- po rozpakowaniu paczki `db-derby-10.15.2.0-bin.zip` do katalogu: `/opt/derby/`
- uruchamiamy narzędzie linii poleceń: `/opt/derby/bin/ij`

```
ij> connect 'jdbc:derby:./data';
ij> set schema=sa
ij> select * from Books;
```

```
conn = DriverManager.getConnection("jdbc:derby:./data;create=true", "SA", "");
```

- po rozpakowaniu paczki `db-derby-10.15.2.0-bin.zip` do katalogu: `/opt/derby/`
- uruchamiamy narzędzie linii poleceń: `/opt/derby/bin/ij`

```
ij> connect 'jdbc:derby:./data';
ij> set schema=sa
ij> select * from Books;
```

- w przykładzie baza była utworzona wywołaniem:

```
conn = DriverManager.getConnection("jdbc:derby:./data;create=true", "SA", "");
```

```java
package pap;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.apache.commons.io.FileUtils;

public class HelloJavaDb {
  String databaseFolder = "/home/jmy/Private/2020Z/JDBC/code/derby/derbysample/data";
  Connection conn;

  public static void main(String[] args) throws SQLException, IOException {
    HelloJavaDb app = new HelloJavaDb();

    app.connectionToDerby();
    app.normalDbUsage();
  }

  public void connectionToDerby() throws SQLException, IOException {
    // drop table
    if (Files.isDirectory(Paths.get(databaseFolder)))
      FileUtils.deleteDirectory(new File(databaseFolder));
    // ----------------------------------------
    // URL format is
    // jdbc:derby:<local directory to save data>
    // ----------------------------------------
    String dbUrl = String.format("jdbc:derby:%s;create=true", databaseFolder);
    conn = DriverManager.getConnection(dbUrl);
  }

  public void normalDbUsage() throws SQLException, IOException {
    Statement stmt = conn.createStatement();
```

```
40
41      // create tables
42      stmt.executeUpdate("CREATE TABLE Persons (" +
43          "PersonID int NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1)," +
44          "LastName varchar(255) NOT NULL,"+
45          "FirstName varchar(255)," +
46          "CONSTRAINT PK_Person PRIMARY KEY (PersonID)" + ")"
47      );
48      stmt.executeUpdate("CREATE TABLE Orders (" +
49          "OrderID int NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1)," +
50          "OrderNumber int NOT NULL," +
51          "PersonID int,"+
52          "PRIMARY KEY (OrderID)," +
53          "CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)" +
54          "REFERENCES Persons(PersonID)" + ")
55      ");
56
57      // insert data
58      for(var sql: new String[]{
59        "INSERT INTO Persons (FirstName, LastName) VALUES ('Jeremy', 'Clarkson')",
60        "INSERT INTO Persons (FirstName, LastName) VALUES ('James', 'May')",
61        "INSERT INTO Persons (FirstName, LastName) VALUES ('Richard', 'Hammond')",
62        "INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020003, 1)",
63        "INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020010, 3)",
64        "INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020014, 1)",
65        "INSERT INTO Orders (OrderNumber, PersonID) VALUES (2020034, 1)"
66      }){
67        stmt.executeUpdate(sql);
68      }
69
70      // query
71      ResultSet rs = stmt.executeQuery("SELECT * FROM Persons");
72
73      // print out query result
74      while (rs.next()) {
75        System.out.printf("%d\t%s\t%s\n", rs.getInt("PersonID"), rs.getString("FirstName"),
          ↪  rs.getString("LastName"));
76      }
77
78      rs = stmt.executeQuery(
79      "SELECT Orders.OrderNumber, Persons.FirstName, Persons.LastName "+
80      "FROM Orders "+
81      " INNER JOIN Persons ON Orders.PersonID = Persons.PersonID");
```

```
82      // print out query result
83      while (rs.next()) {
84        System.out.printf("%d\t%s\t%s\n", rs.getInt("OrderNumber"), rs.getString("FirstName"),
          ↪  rs.getString("LastName"));
85      }
86
87      System.out.printf("the end\n");
88      conn.close();
89    }
90  }
```

- instalacja pakietu na Ubuntu 20.04

```
sudo apt install mysql-server
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń

```
1 sudo mysql_secure_installation
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
  - instalacja VALIDATE PASSWORD PLUGIN - wymusza reguły dla hasła

```
1 sudo mysql_secure_installation
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
  - instalacja VALIDATE PASSWORD PLUGIN - wymusza reguły dla hasła
  - ustawienie hasła dla użytkownika root MySQL

```
1 sudo mysql_secure_installation
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
  - instalacja VALIDATE PASSWORD PLUGIN - wymusza reguły dla hasła
  - ustawienie hasła dla użytkownika root MySQL
  - usunięcie użytkownika anonimowego

```
1 sudo mysql_secure_installation
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
  - instalacja VALIDATE PASSWORD PLUGIN - wymusza reguły dla hasła
  - ustawienie hasła dla użytkownika root MySQL
  - usunięcie użytkownika anonimowego
  - ograniczenie dostępu do lokalnej maszyny

```
1 sudo mysql_secure_installation
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
  - instalacja VALIDATE PASSWORD PLUGIN - wymusza reguły dla hasła
  - ustawienie hasła dla użytkownika root MySQL
  - usunięcie użytkownika anonimowego
  - ograniczenie dostępu do lokalnej maszyny
  - usunięcie bazy testowej

```
1 sudo mysql_secure_installation
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
- Zmiana sposobu autentykacji roota
  - Logowanie do bazy

```
1 sudo mysql
```

- W systemach Ubuntu z MySQL 5.7 (i nowszym) użytkownik root jest domyślnie uwierzytelniany przez wtyczkę auth_socket.
- Wtyczka auth_socket uwierzytelnia użytkowników, którzy łączą się z hosta lokalnego poprzez plik gniazda Unix.
- Oznacza to, że nie możesz uwierzytelnić się jako root, podając hasło.

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
- Zmiana sposobu autentykacji roota
  - Logowanie do bazy
  - zmiana metody uwierzytelniania z auth_socket na mysql_native_password

```
1 ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'very_strong_password';
2 FLUSH PRIVILEGES;
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
- Zmiana sposobu autentykacji roota
- utwórz nowego użytkownika administracyjnego z dostępem do wszystkich baz danych

```
1 CREATE USER 'administrator'@'localhost' IDENTIFIED BY '<very strong password>';
2 GRANT ALL ON *.* TO 'administrator'@'localhost';
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
- Zmiana sposobu autentykacji roota
- utwórz nowego użytkownika administracyjnego z dostępem do wszystkich baz danych
- instalacja phpmyadmin (apache)
  - gdy nie uruchomiliśmy wtyczki Validate Password

```
1 sudo apt install phpmyadmin
```

- instalacja pakietu na Ubuntu 20.04
- uruchamiamy skrypt `mysql_secure_installation` który zwiększa poziom zabezpieczeń
- Zmiana sposobu autentykacji roota
- utwórz nowego użytkownika administracyjnego z dostępem do wszystkich baz danych
- instalacja phpmyadmin (apache)
  - gdy nie uruchomiliśmy wtyczki Validate Password
  - gdy uruchomiliśmy wtyczkę Validate Password - trzeba ją wyłączyć

```
sudo mysql      #  or: mysql -u root -p
mysql>UNINSTALL COMPONENT "file://component_validate_password";
mysql>exit
sudo apt install phpmyadmin
sudo mysql      #  or: mysql -u root -p
mysql>INSTALL COMPONENT "file://component_validate_password";
mysql>exit
```

```
CREATE USER 'pap20Z'@'localhost' IDENTIFIED WITH caching_sha2_password BY '***';
GRANT ALL PRIVILEGES ON *.* TO 'pap20Z'@'localhost' WITH GRANT OPTION;
ALTER USER 'pap20Z'@'localhost' REQUIRE NONE
  WITH MAX_QUERIES_PER_HOUR 0
  MAX_CONNECTIONS_PER_HOUR 0
  MAX_UPDATES_PER_HOUR 0
  MAX_USER_CONNECTIONS 0;
CREATE DATABASE IF NOT EXISTS `pap20Z`;
GRANT ALL PRIVILEGES ON `pap20Z`.* TO 'pap20Z'@'localhost';
```