



# JDBC



# Prowadzący

Bartosz Andreatto  
**(bandreatto@gmail.com)**

programista – Bank Pekao S.A.

<https://pl.linkedin.com/in/bartosz-andreatto-02b03413a>



# Plan zajęć

- JDBC – architektura i komponenty
- Nawiązywanie i zamykanie połączeń z bazą danych
- Wykonywanie zapytań
- Wykonywanie modyfikacji danych
- Wykonywanie modyfikacji struktury danych
- Pobieranie meta-danych
- Prekomplilowane polecenia
- Wywoływanie procedur składowanych
- Aktualizowanie wsadowe
- Podsumowanie



# JDBC

- **Java DataBase Connectivity** - łącze do baz danych w języku Java
- **Interfejs programowania** opracowany w 1996 r. przez Sun Microsystems, umożliwiający **niezależnym od platformy aplikacjom napisanym w języku Java porozumiewanie się z bazami danych za pomocą języka SQL**. Interfejs ten jest odpowiednikiem standardu ODBC opracowanego przez SQL Access Group.
- Środowisko Java Platform, Standard Edition **zawiera API JDBC**, natomiast użytkownik **musi uzyskać specjalny sterownik JDBC do swojej bazy danych**. Możliwe jest łączenie się z ODBC przez specjalne sterowniki, tłumaczące odwołania JDBC na komendy ODBC.



# JDBC

Wymaga:

- **Bazy danych**
- **Sterownika** – dostarczany przez producenta bazy danych



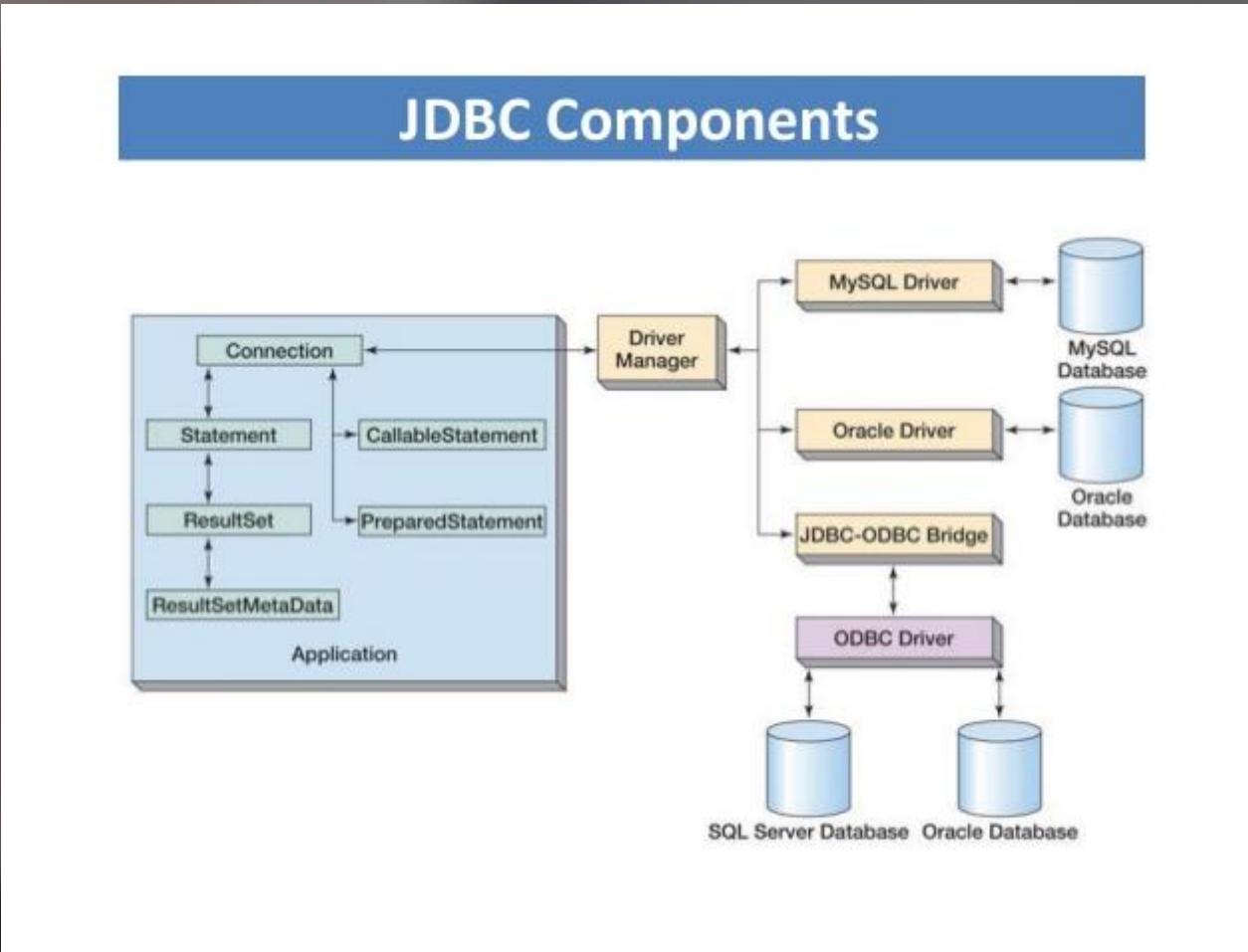
# JDBC

Główne funkcjonalności, jakie JDBC realizuje to:

- możliwość **nawiązywania połączeń** z bazą danych
- **wysyłanie zapytań** i instrukcji do bazy danych
- **pobieranie i przetwarzanie wyników** zapytań SQL

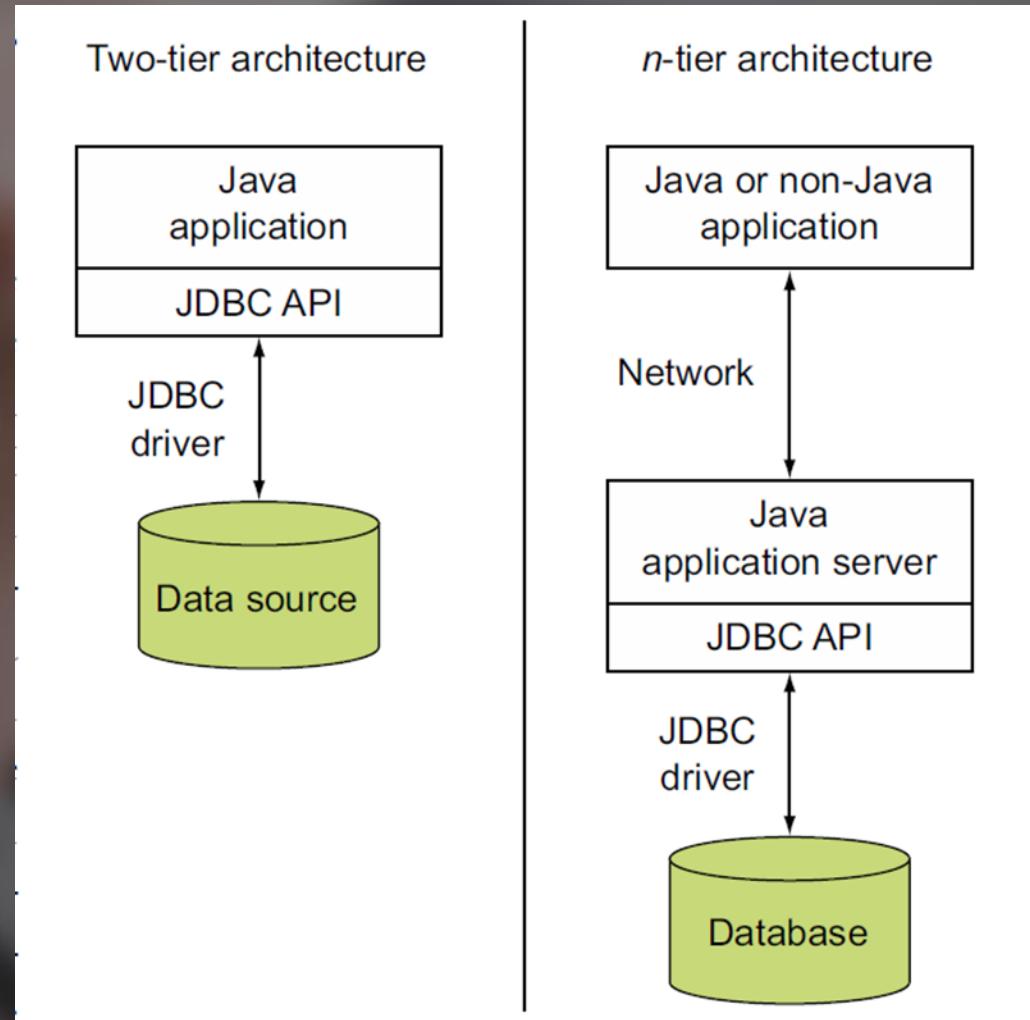


# JDBC - architektura





# JDBC - architektura





# JDBC – podstawowe komponenty

- **JDBC API** - jest to najważniejsza część JDBC, która pozwala, **za pomocą wywoływanego metod z poziomu kodu, wywoływać zapytania na bazie danych**
- **JDBC Driver Manager** - definiuje **w jaki sposób nawiązać połączenie z bazą danych.**
- **JDBC Test Suite** - jest zbiorem testów pomagających tworzenie sterownika JDBC, który jest elementem niezbędnym do nawiązywania połączenia.
- **JDBC-ODBC Bridge** - implementacja wykorzystująca bridge w celu umożliwienia wykorzystania JDBC API za pomocą sterowników ODBC (Open DataBase Connectivity)

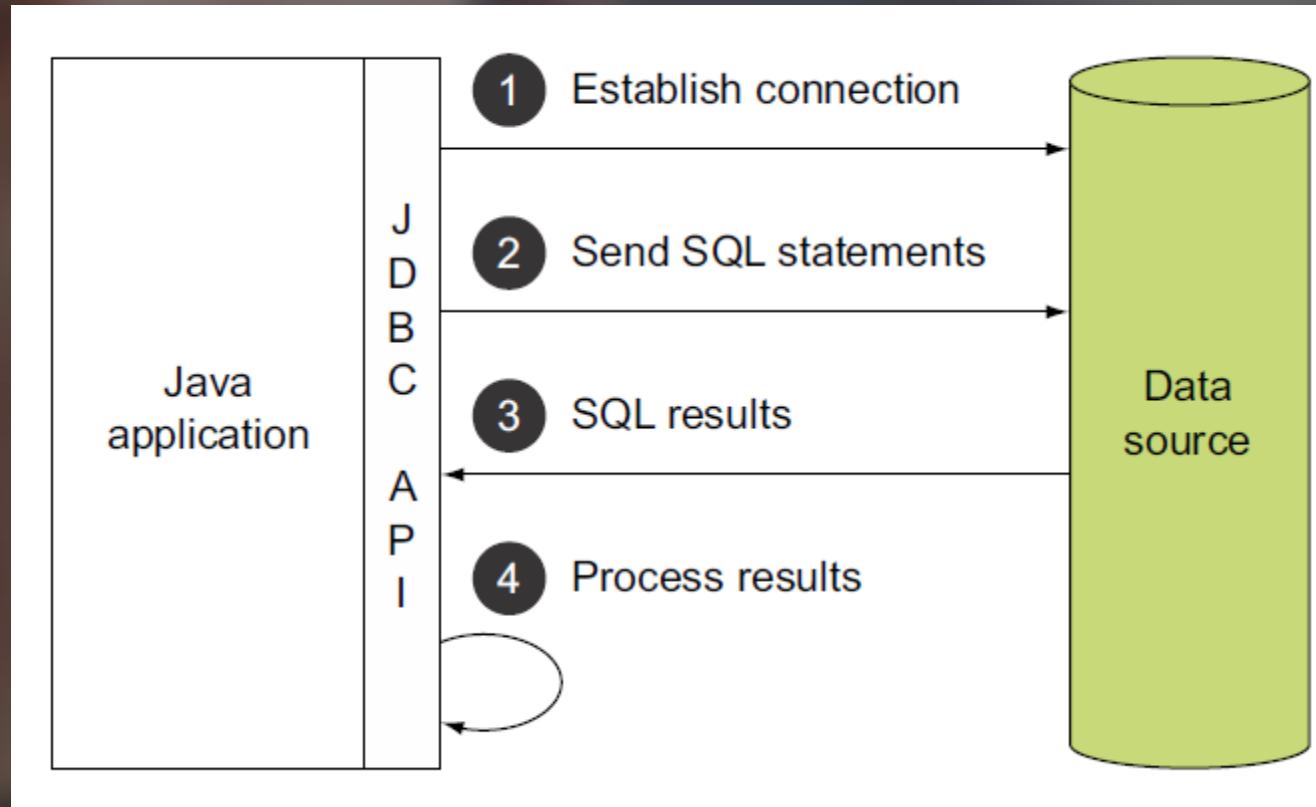


# JDBC – podstawowe komponenty

- **Driver** – wie jak uzyskać połączenie z bazą danych
- **Connection** – wie w jaki sposób komunikować się z bazą danych
- **Statement** – wie w jaki sposób wykonać polecenie SQL
- **ResultSet** – wie co zostało zwrócone w wyniku zapytania SELECT

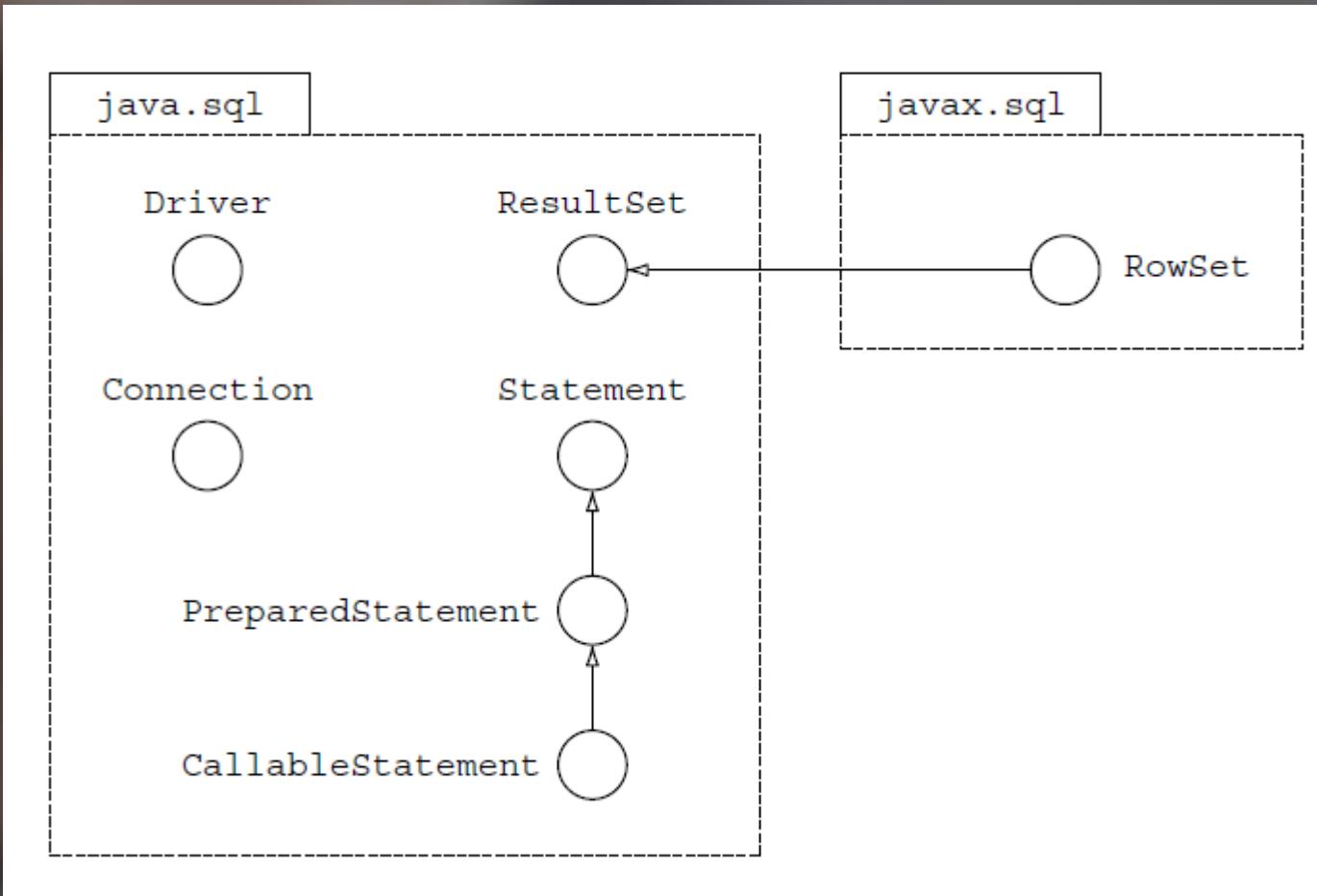


# JDBC – podstawowe komponenty





# JDBC – podstawowe komponenty





# JDBC – nawiązywanie połączenia z bazą danych

W celu wykonania jakiejkolwiek interakcji z bazą danych **musimy najpierw się z nią połączyć**. Każdy serwer bazodanowy (np. MySQL, MariaDB, Oracle DB czy Microsoft SQL Server) **posiada nieco inną implementację i w celu nawiązania połączenia do każdego z nich za pomocą takiej samej metody, potrzebna jest warstwa łącząca**. Taką warstwą jest **Driver**, czyli sterownik, który jest specyficzny dla konkretnej implementacji bazy danych.



# JDBC – nawiązywanie połączenia z bazą danych

Aby móc korzystać z JDBC, **potrzebujemy konkretnej implementacji sterownika** znajdującego się na classpath.

Na przykład w celu połączenia się do bazy MySQL 8.x, w projekcie wykorzystującym mavena, w pliku **pom.xml** w sekcji dependencies powinniśmy wykorzystać następującą zależność:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.20</version>
</dependency>
```

Zazwyczaj informacje o sterownikach, możemy znaleźć na stronie dostawcy bazy danych, np. <https://dev.mysql.com/downloads/connector/j/>.



# JDBC – nawiązywanie połączenia z bazą danych

Sposoby uzyskiwania połączenia w oparciu o obiekty:

- **DriverManager**
- **DataSource**

Wymagany zestaw danych:

- **nazwa użytkownika**
- **hasło użytkownika**
- **adres url bazy danych**



# JDBC – nawiązywanie połączenia z bazą danych

Budowa adresu URL bazy danych:

- **[protokół]:[dostawca db]:[szczegóły połączenia]**

Przykłady:

- **MySQL: jdbc:mysql://localhost:3306/sys**
- Oracle: jdbc:oracle:thin:@localhost:1521:xe
- PostgreSQL: jdbc:postgresql://lohalhost:5432/acounting



# JDBC – nawiązywanie połączenia z bazą danych

```
public static Connection getConnection
    (String url) throws SQLException
public static Connection getConnection
    (String url, Properties info) throws SQLException
public static Connection getConnection
    (String url, String user, String pwd) throws SQLException
```



# JDBC – nawiązywanie połączenia z bazą danych

```
public static Connection getConnection() throws SQLException {
    return DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
}
```

```
public static Connection getConnectionBasedOnProperties() throws SQLException {
    Properties properties = new Properties();
    properties.setProperty("user", DB_USER);
    properties.setProperty("password", DB_PASSWORD);

    return DriverManager.getConnection(DB_URL, properties);
}
```

```
public static Connection getConnectionBasedOnURLParameters() throws SQLException {
    String parametrizedDbUrl = DB_URL + "&user=" + DB_USER + "&password=" + DB_PASSWORD;
    log.info("Parametrized database URL: " + parametrizedDbUrl);

    return DriverManager.getConnection(parametrizedDbUrl);
}
```

```
private static final String DB_URL = "jdbc:mysql://localhost:3306/sda?serverTimezone=UTC";
```



# JDBC – nawiązywanie połączenia z bazą danych

Wcześniejsze wersje wymagały:

```
static {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
    } catch (InstantiationException | IllegalAccessException | ClassNotFoundException exception) {
        log.error("Can't load jdbc driver class!", exception);
    }
}
```



# JDBC – nawiązywanie połączenia z bazą danych

```
public static DataSource getDataSource() {  
    MysqlDataSource dataSource = new MysqlDataSource();  
    Properties properties = LoadProperties();  
  
    dataSource.setUrl(properties.getProperty("mysql.url"));  
    dataSource.setUser(properties.getProperty("mysql.username"));  
    dataSource.setPassword(properties.getProperty("mysql.password"));  
  
    return dataSource;  
}
```

```
try (Connection connection = getConnectionBasedOnURLParameters()) {  
    log.info("Connected to database successfully!");  
    log.info("Connection: " + connection);  
    log.info("Database name: " + connection.getCatalog());  
} catch (SQLException exception) {  
    log.error("Can't obtain db connection!", exception);  
    throw new RuntimeException("Can't obtain db connection!", exception);  
}
```



# JDBC – wykonywanie zapytań

Interakcja z bazą danych realizowana jest za pomocą obiektu klasy **Statement**, który umożliwia **wysyłanie zapytań SQL oraz konsumowanie odpowiedzi**. Oprócz bazowej klasy Statement istnieją dwie klasy pochodne:

- **PreparedStatement**
- **CallableStatement**

Musimy pamiętać, że tworzenie obiektu Statement, **wymaga jego zamknięcia, stąd w praktyce często metodę createStatement na obiekcie Connection wywołujemy korzystając z try z zasobem.**



# Statement

Mając nawiązane połączenie do bazy danych i stworzony obiekt Statement, możemy za jego pomocą **wywołać zapytanie SQL**. Do wywołania takiego zapytania możemy wykorzystać przeciążenia metod:

- **execute** – metoda może wykonać zapytanie typu SELECT, UPDATE, DELETE, INSERT, CREATE
- **executeUpdate** – służy głównie do wykonywania zapytań typu UPDATE, DELETE, INSERT
- **executeQuery** - służy do pobierania danych z bazy danych (tzn. najczęściej jest wykorzystywana do wywoływania SELECT)



# Statement

Method	DELETE	INSERT	SELECT	UPDATE
stmt.execute()	Yes	Yes	Yes	Yes
stmt.executeQuery()	No	No	Yes	No
stmt.executeUpdate()	Yes	Yes	No	Yes

Method	Return Type	What Is Returned for SELECT	What Is Returned for DELETE/INSERT/UPDATE
stmt.execute()	boolean	true	false
stmt.executeQuery()	ResultSet	The rows and columns returned	n/a
stmt.executeUpdate()	int	n/a	Number of rows added/changed/removed



# ResultSet

Kolejnym ważnym obiektem, który **reprezentuje tabelę będącą wynikiem** wykonanego zapytania jest instancja **ResultSet** (który również implementuje interfejs **AutoCloseable** ale jest on **automatycznie zamykany podczas zamykania instancji Statement**). Opiera się on na **kursorze**, który możemy przesuwać po tabeli wynikowej, np. do kolejnego wiersza możemy dostać się za pomocą wywołania metody **next()**. Wartość kolumny możemy pobrać za pomocą odpowiedniej metody **getX**, gdzie X jest typem kolumny (np. String, Boolean czy Byte). Co ważne, **pierwsza kolumna ma indeks 1.**

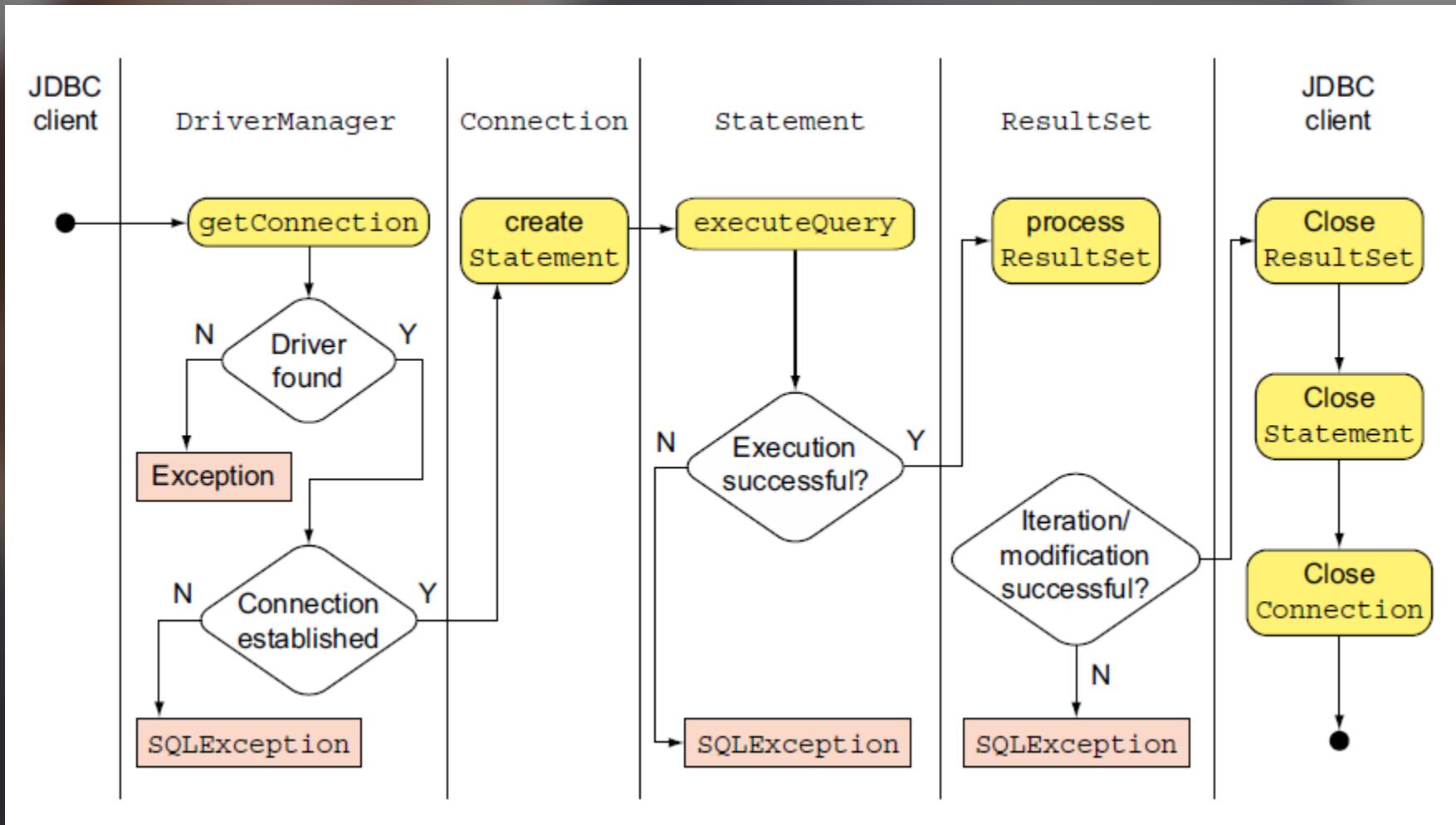


# ResultSet

Często wykorzystywane metody na obiekcie typu ResultSet to:

- **next** – przesuwa kursor do następnego rekordu
- **previous** – przesuwa kursor do poprzedniego rekordu
- **first** – przesuwa kursor do pierwszego rekordu
- **last** – przesuwa kursor do ostatniego rekordu

# Wykonywanie zapytań





# Wykonywanie zapytań

```
String selectQuery = "SELECT * FROM db_table";  
Statement selectStatement = connection.createStatement();  
ResultSet resultSet = selectStatement.executeQuery(selectQuery);  
doSomethingWith(resultSet);  
selectStatement.close();
```

```
String insertQuery = "INSERT INTO db_table (name, value) VALUES ('test', 42)";  
Statement insertStatement = connection.createStatement();  
int resultCode = insertStatement.executeUpdate(insertQuery);  
doSomethingWith(resultCode);  
insertStatement.close();
```



# ResultSet

```
while(resultSet.next()) {  
    Long id = resultSet.getLong(columnLabel: "id");  
    String text = resultSet.getString(columnLabel: "text");  
    Integer number = resultSet.getInt(columnLabel: "number");  
    Date date = resultSet.getDate(columnLabel: "date");  
    Boolean flag = resultSet.getBoolean(columnLabel: "flag");  
    doSomethingWith(id, text, number, date, flag);  
}  
resultSet.close();
```



# ResultSet

Initial position	id integer	name character varying(255)	num_acres numeric
rs.next() true	1	African Elephant	7.5
rs.next() true	2	Zebra	1.2
rs.next() false			



# ResultSet

Method Name	Return Type	Example Database Type
getBoolean	boolean	BOOLEAN
getDate	java.sql.Date	DATE
getDouble	double	DOUBLE
getInt	int	INTEGER
getLong	long	BIGINT
getObject	Object	Any type
getString	String	CHAR, VARCHAR
getTime	java.sql.Time	TIME
getTimeStamp	java.sql.Timestamp	TIMESTAMP



# ResultSet

```
Statement stmt = conn.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_READ_ONLY);
```

	<code>beforeFirst()</code>	<code>id</code>	<code>name</code>	<code>num_acres</code>
		integer	character varying(255)	numeric
<code>first()</code>		1	African Elephant	7.5
<code>last()</code>		2	Zebra	1.2
<code>afterLast()</code>				

	<code>absolute(0)</code>	<code>id</code>	<code>name</code>	<code>num_acres</code>
		integer	character varying(255)	numeric
<code>absolute(1)</code>		1	African Elephant	7.5
<code>absolute(2)</code>		2	Zebra	1.2
<code>absolute(3)</code>				

	<code>absolute(-3)</code>	<code>id</code>	<code>name</code>	<code>num_acres</code>
		integer	character varying(255)	numeric
<code>absolute(-2)</code>		1	African Elephant	7.5
<code>absolute(-1)</code>		2	Zebra	1.2



# ResultSet

Method	Description	Requires Scrollable ResultSet
boolean absolute(int rowNum)	Move cursor to the specified row number	Yes
void afterLast()	Move cursor to a location immediately after the last row	Yes
void beforeFirst()	Move cursor to a location immediately before the first row	Yes
boolean first()	Move cursor to the first row	Yes
boolean last()	Move cursor to the last row	Yes
boolean next()	Move cursor one row forward	No
boolean previous()	Move cursor one row backward	Yes
boolean relative(int rowNum)	Move cursor forward or backward the specified number of rows	Yes



# ResultSet

Method	Description	Requires Scrollable ResultSet
boolean absolute(int rowNum)	Move cursor to the specified row number	Yes
void afterLast()	Move cursor to a location immediately after the last row	Yes
void beforeFirst()	Move cursor to a location immediately before the first row	Yes
boolean first()	Move cursor to the first row	Yes
boolean last()	Move cursor to the last row	Yes
boolean next()	Move cursor one row forward	No
boolean previous()	Move cursor one row backward	Yes
boolean relative(int rowNum)	Move cursor forward or backward the specified number of rows	Yes



# Wykonywanie zapytań - przykłady

Założmy, że tworzymy w bazie następującą tabelę wraz z danymi:

```
1 CREATE TABLE Persons (
2     PersonID int,
3     LastName varchar(255),
4     FirstName varchar(255),
5     Address varchar(255),
6     City varchar(255)
7 );
8
9     INSERT INTO Persons VALUES (1, 'Andrzejewski', 'Andrzej', 'Nowaka 1', 'Warszawa');
10    INSERT INTO Persons VALUES (1, 'Adamski', 'Adam', 'Adamicka 1', 'Adamowo');
```



# Wykonywanie zapytań - przykłady

Zapytanie na takiej tabeli możemy wykonać, na przykład wykorzystując metodę executeQuery:

```
1 public static void main(String[] args) throws SQLException {
2
3     try (Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/sda", "root", "my-secret-pw");
4          Statement statement = connection.createStatement()) {
5
6         final ResultSet resultSet = statement.executeQuery("SELECT * FROM Persons");
7         while (resultSet.next()) {
8             System.out.println("PersonID " + resultSet.getInt(1));
9             System.out.println("LastName " + resultSet.getString(1));
10            System.out.println("FirstName " + resultSet.getString(2));
11            System.out.println("Address " + resultSet.getString(3));
12            System.out.println("City " + resultSet.getString(4));
13        }
14    }
15 }
```



# Wykonywanie zapytań - przykłady

```
1 try (Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/sda", "root", "my-secret-pw");
2     Statement statement = connection.createStatement()) {
3     boolean isResultSetReturned = statement.execute("create table Book (Id INT PRIMARY KEY, Title VARCHAR(32) NOT NULL, Type VARCHAR(32), Description VARCHAR(50))");
4     System.out.println(isResultSetReturned); // false -> pierwszy wynikiem NIE jest ResultSet
5 }
```

```
1 try (Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/sda", "root", "my-secret-pw");
2     Statement statement = connection.createStatement()) {
3
4     int rowCount = statement.executeUpdate("insert into Book values(1, 'Java beginner', 'Programowanie', 'Programowanie w javie od podstaw')");
5     System.out.println(rowCount); // 1
6 }
```



# PreparedStatement

**PreparedStatement** jest reprezentacją **szkieletu zapytania SQL**.

Posiada ona tzw. **placeholdery** tzn. miejsca, które **można zastąpić dowolną wartością**. W zapytaniu reprezentowane są poprzez znak **?**.

Wartość takiego placeholdera możemy zastąpić odpowiednią metodą **setX**, w zależności od używanego typu danych, np. **setInt** czy **setString**. Podczas wykorzystywania tych metod, **musimy podać indeks kolumny (pierwsza kolumna ma indeks 1)**.



# PreparedStatement

```
String queryTemplate =  
    "SELECT * FROM db_table WHERE string = ? AND number = ? AND date = ?";  
PreparedStatement statement = connection.prepareStatement(queryTemplate);  
statement.setString( parameterIndex: 1, stringValue);  
statement.setInt( parameterIndex: 2, intValue);  
statement.setDate( parameterIndex: 3, dateValue);  
boolean hasResultSet = statement.execute();  
doSomethingWith(hasResultSet, statement);  
statement.close();
```



# CallableStatement

**CallableStatement** jest kolejnym typem obiektu Statement, który służy do wykonywania procedur (zlokalizowanych po stronie serwera bazodanowego). CallableStatement, podobnie jak PreparedStatement, **pozwala na korzystanie placeholderów**.



# CallableStatement

```
CREATE FUNCTION getDob(emp_name VARCHAR(50)) RETURNS DATE
BEGIN
declare dateOfBirth DATE;
select DOB into dateOfBirth from EMP where Name = emp_name;
return dateOfBirth;
END
```

```
//Registering the Driver
DriverManager.registerDriver(new com.mysql.jdbc.Driver());

//Getting the connection
String mysqlUrl = "jdbc:mysql://localhost/sampleDB";
Connection con = DriverManager.getConnection(mysqlUrl, "root", "password");
System.out.println("Connection established.....");

//Preparing a CallableStatement
CallableStatement cstmt = con.prepareCall("{? = call getDob(?)}");

cstmt.registerOutParameter(1, Types.DATE);
cstmt.setString(2, "Amit");
cstmt.execute();

System.out.print("Date of birth: "+cstmt.getDate(1));
}
```



# Podsumowanie

## Zalety:

- Duża kontrola nad wykonywanymi zapytaniami

## Wady:

- Powiązanie z konkretną bazą danych
- Dużo powtarzalnego kodu
- Konieczność zarządzania wyjątkami i połączeniami



# ĆWICZENIA



# Dziękuję