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

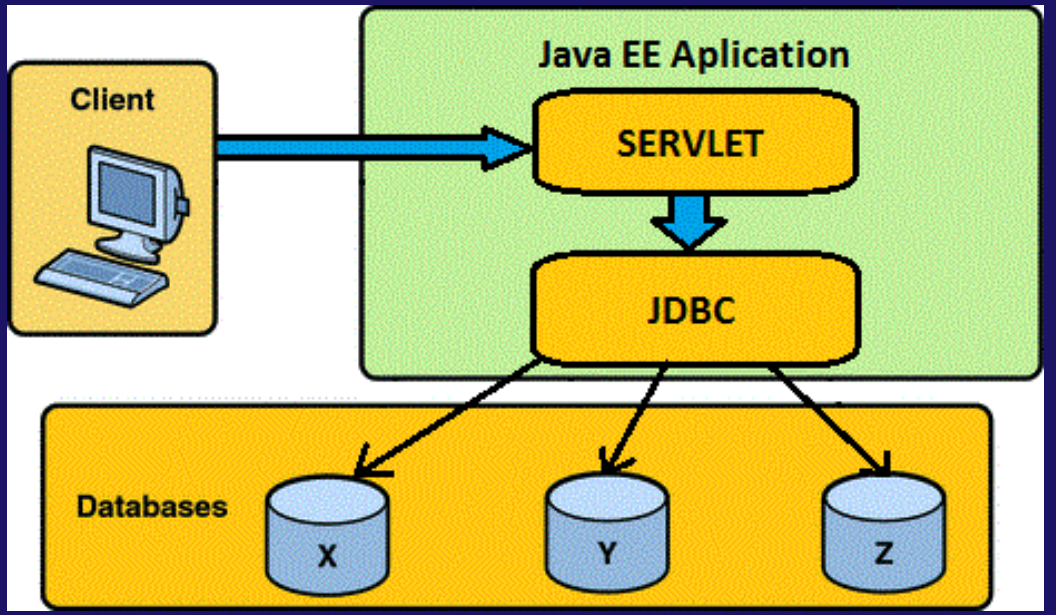
https://github.com/umcspro/rent_car_stud

JDBC

Java Database Connectivity

JDBC

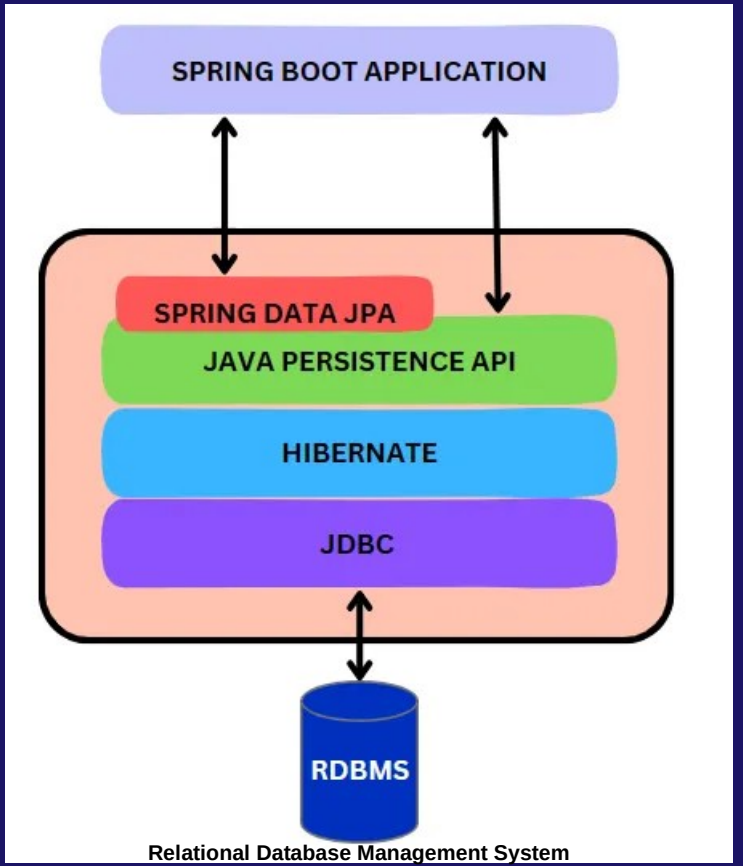
Java Database Connectivity jest to interfejs pozwalający na ustanowienie połączenia do relacyjnej bazy danych z poziomu aplikacji napisanej w Javie.



https://javastart.pl/baza-wiedzy/java-ee/jdbc-podstawy-pracy-z-baza-danych

JDBC

W aplikacjach pisanych w Spring Framework i Hibernate JDBC jest jedną z warstw.



https://medium.com/@aprayush20/spring-data-jpa-with-spring-boot-9e3b261e4cc7

Komponenty

JDBC

DriverManager

Klasa, która zarządza listą sterowników baz danych. Umożliwia aplikac
uzyskanie połączenia z bazą danych poprzez wywołanie metody
getConnection.

Connection

Interfejs reprezentujący połączenie z bazą danych. Umożliwia
wykonywanie poleceń SQL, zarządzanie transakcjami i
uzyskiwanie metadanych bazy danych.

Statement

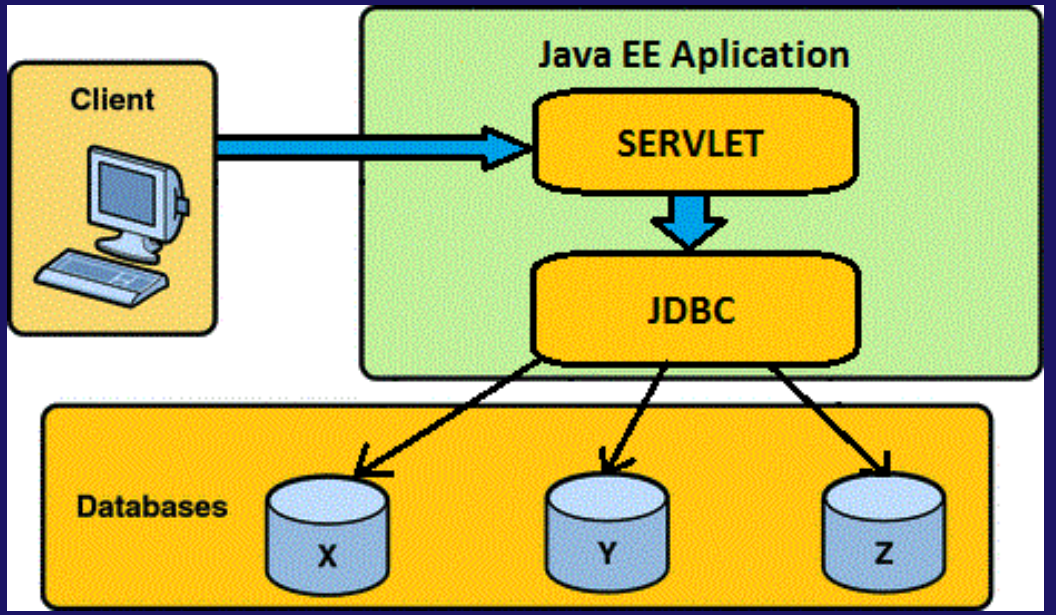
Interfejs służący do wykonywania zapytań SQL. Istnieją różne
typy obiektów Statement, takie jak Statement,
PreparedStatement i CallableStatement, które odpowiadają za
wykonanie zapytań, przygotowane zapytania i wywołania
procedur składowanych.

ResultSet

Interfejs reprezentujący zbiór wyników zapytania SQL. Umożliwia
iterowanie po wynikach i odczytywanie wartości poszczególnych
kolumn.

Baza danych

Przed prześledzeniem działania komponentów JDBC należy wybrać bazę i sterownik.



https://javastart.pl/baza-wiedzy/java-ee/jdbc-podstawy-pracy-z-baza-danych

Oto darmowe serwisy:
https://www.db4free.net/index.php?language=pl
https://www.elephantsql.com/

Baza danych

Konektory MySQL oraz PostgreSQL:

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.7.3</version>
</dependency>
```

Baza danych

Przykładowa baza danych z zajęć wypełniona danymi:

Baza danych

```
CREATE TABLE `tuser` (
  `login` varchar(64) NOT NULL,
  `password` varchar(255) NOT NULL,
  `rentedPlate` varchar(64) DEFAULT '000',
  `role` enum('USER','ADMIN') NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;
```

```
INSERT INTO `tuser` (`login`, `password`, `rentedPlate`, `role`) VALUES('lukasz',
'6a01d2ff826b812897ab3dec11e939779d3b06dc3625c3377bd4ae9639e8a9bd', 'Lu1234', 'ADMIN');
INSERT INTO `tuser` (`login`, `password`, `rentedPlate`, `role`) VALUES('student',
'ad454dc5db203e4280041fcd250c3de1188cf66613d03a8fc6f0eadc3d1bec97', NULL, 'USER');
```

Baza danych

```sql
CREATE TABLE `tvehicle` (
  `plate` varchar(64) NOT NULL,
  `brand` varchar(255) NOT NULL,
  `model` varchar(255) NOT NULL,
  `year` int(6) NOT NULL,
  `price` decimal(6,2) NOT NULL,
  `category` varchar(64) DEFAULT NULL,
  `rent` tinyint(1) NOT NULL DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;
```

```sql
INSERT INTO `tvehicle` (`plate`, `brand`, `model`, `year`, `price`, `category`, `rent`)
VALUES('Lu1000', 'Honda', 'CBR1000RR-R Fireblade SP', 2023, 500.00, 'A', 0);
INSERT INTO `tvehicle` (`plate`, `brand`, `model`, `year`, `price`, `category`, `rent`)
VALUES('Lu1234', 'Audi', 'A4', 2021, 400.00, NULL, 1);
```

Baza danych

```
ALTER TABLE `tuser`
  ADD PRIMARY KEY (`login`) USING BTREE,
  ADD KEY `fk_user_vehicle` (`rentedPlate`);

ALTER TABLE `tvehicle`
  ADD PRIMARY KEY (`plate`) USING BTREE;

ALTER TABLE `tuser`
  ADD CONSTRAINT `fk_user_vehicle` FOREIGN KEY (`rentedPlate`) REFERENCES `tvehicle` (`plate`);
```

DriverManager

1. Rejestracja sterowników
2. Nawiązywanie połączeń
3. Zarządzanie sterownikami

Nawiązywanie połączenia wygląda bardzo prosto:

```
DriverManager.getConnection(url, user, password);
```

DriverManager

Połączenie może nam zwracać własna klasa singleton:

```
public class DatabaseManager {
    private static DatabaseManager instace = null;
    private final String url;
    private final String user;
    private final String password;
    public static DatabaseManager getInstance() {
        if(DatabaseManager.instace==null){
                DatabaseManager.instace = new DatabaseManager();
        }
        return DatabaseManager.instace;
    }
    private DatabaseManager(){
        this.url = "jdbc:mysql://localhost:3306/spring2024";
        this.user = "root";
        this.password = "***";
    }

    public Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url, user, password);
    }
}
```

```
DriverManager.getConnection(url, user, password)
```

Connection

1.Wykonywanie Zapytań SQL:

Metody do tworzenia poleceń:
Statement,
PreparedStatement,
CallableStatement,
do wykonywania zapytań SQL, takich jak SELECT, INSERT, UPDATE, DELETE oraz wywoływanie procedur składowanych.

2.Zarządzanie Transakcjami

Manualnie:
connection.setAutoCommit(false)
Zatwiedzanie:
connection.commit()
Odrzucanie:
connection.rollback().

3.Zamykanie połączenia

connection.close().

Connection

```java
public User getUser(String login) {
    User user = null;
    Connection connection =null;
    ResultSet rs = null;
    try{
        Connection = databaseManager.getConnection();
        connection.createStatement();
        PreparedStatement preparedStatement = connection.prepareStatement(GET_USER_SQL);
        preparedStatement.setString(1, login);
        rs = preparedStatement.executeQuery();
        if(rs.next()) {
            user = new User(
                rs.getString("login"),
                rs.getString("password"),
                User.Role.valueOf(rs.getString("role")),
                rs.getString("rentedPlate")
            );
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    finally {
            //zamkniecia
    }
    return user;
}
```

```java
private static String GET_USER_SQL = "SELECT * FROM tuser WHERE login LIKE ?";
```

Connection

```
@Override
public void removeUser(String login) {

    try (Connection conn =
databaseManager.getConnection();
         PreparedStatement stmt =
conn.prepareStatement(DELETE_SQL)) {
        stmt.setString(1, login);
        int changed = stmt.executeUpdate();
        if (changed > 0) {
            System.out.println("User usunięty.");
        } else {
            System.out.println("Nie usuniety");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```
private static String DELETE_SQL = "DELETE FROM tuser WHERE login LIKE ?";
```

Connection

```
public boolean addVehicle(Vehicle vehicle) {
    try (Connection conn = manager.getConnection();
         PreparedStatement stmt = conn.prepareStatement(INSERT_SQL)) {

        stmt.setString(1, vehicle.getBrand());
        stmt.setString(2, vehicle.getModel());
        stmt.setInt(3, vehicle.getYear());
        stmt.setDouble(4, vehicle.getPrice());
        stmt.setString(5, vehicle.getPlate());

        int changed = stmt.executeUpdate();

        if (changed  > 0) {
            System.out.println("Pojazd został pomyślnie dodany.");
            return true;
        } else {
            System.out.println("Nie udało się dodać pojazdu.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

```
private final String INSERT_SQL = "INSERT INTO tvehicle (brand, model, year, price, plate) VALUES (?,?,?,?,?)";
```

Connection

```
public boolean rentVehicle(String plate, String login) {
    Connection conn = null;
    PreparedStatement rentCarStmt = null;
    PreparedStatement updateUserStmt = null;

    try {
        conn = manager.getConnection();
        conn.setAutoCommit(false); // reczny commit

                                rentCarStmt = conn.prepareStatement(RENT_CAR_SQL);
                                rentCarStmt.setString(1, plate);
                                int changed1 =rentCarStmt.executeUpdate();

                                updateUserStmt = conn.prepareStatement(RENT_UPDATE_USER_SQL);
                                updateUserStmt.setString(1, plate);
                                updateUserStmt.setString(2, login);
                                int changed2 =updateUserStmt.executeUpdate();

                                if (changed1 > 0 && changed2 > 0) {
                                    System.out.println("wypozyczono");
                                    conn.commit();
                                } else {
                                    System.out.println("Nie wypożyczono");
                                    conn.rollback();
                                }

    } catch(Exception e) {
        e.printStackTrace();
        if (conn!= null) {
            try {
                conn.rollback(); // Wycofaj transakcję w przypadku błędu
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
    } finally {
        try { if (rentCarStmt != null) rentCarStmt.close(); } catch (Exception e) {};
        try { if (updateUserStmt != null) updateUserStmt.close(); } catch (Exception e) {};
        try { if (conn != null) conn.close(); } catch (Exception e) {};
    }
    return false;
}
```

Connection

```
public boolean rentVehicle(String plate, String login) {
    Connection conn = null;
    PreparedStatement rentCarStmt = null;
    PreparedStatement updateUserStmt = null;

    try {
        conn = manager.getConnection();
        conn.setAutoCommit(false); // reczny commit

                                rentCarStmt = conn.prepareStatement(RENT_CAR_SQL);
                                rentCarStmt.setString(1, plate);
                                int changed1 =rentCarStmt.executeUpdate();

                                updateUserStmt = conn.prepareStatement(RENT_UPDATE_USER_SQL);
                                updateUserStmt.setString(1, plate);
                                updateUserStmt.setString(2, login);
                                int changed2 =updateUserStmt.executeUpdate();

                                if (changed1 > 0 && changed2 > 0) {
                                    System.out.println("wypozyczono");
                                    conn.commit();
                                } else {
                                    System.out.println("Nie wypożyczono");
                                    conn.rollback();
                                }

    } catch(Exception e) {
        e.printStackTrace();
        if (conn!= null) {
            try {
                conn.rollback(); // Wycofaj transakcję w przypadku błędu
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
    } finally {
        try { if (rentCarStmt != null) rentCarStmt.close(); } catch (Exception e) {};
        try { if (updateUserStmt != null) updateUserStmt.close(); } catch (Exception e) {};
        try { if (conn != null) conn.close(); } catch (Exception e) {};
    }
    return false;
}
```

Dziękuję za uwagę!