Notatki - JDBC - cz.2

Spis treści

Wywołajmy w końcu jakąś komendę ☺

Możemy nareszcie przejść do faktycznego wywołania jakiegoś zapytania SQL.

Zacznijmy od QUERY typu INSERT, gdyż jest to pierwszy rodzaj zapytania jakiego będziemy potrzebowali aby zasilić tabelę danymi. Mamy 3 podstawowe rodzaje metod stosowanych przy interface Statement aby wykonywać zapytania SQL. Jest to metoda execute(), executeQuery() i executeUpdate().

Dane do przykładów

W kolejnych przykładach będę odnosił się cały czas do struktury tabel, która jest pokazana poniżej. Będziemy działać na tabelach: CUSTOMER, PRODUCER, PRODUCT, PURCHASE, OPINION.

Tabela CUSTOMER

```
CREATE TABLE CUSTOMER(
  ID              INT         NOT NULL,
  USER_NAME       VARCHAR(32) NOT NULL,
  EMAIL           VARCHAR(32) NOT NULL,
  NAME            VARCHAR(32) NOT NULL,
  SURNAME         VARCHAR(32) NOT NULL,
  DATE_OF_BIRTH   DATE,
  TELEPHONE_NUMBER VARCHAR(64),
  PRIMARY KEY (ID),
  UNIQUE (USER_NAME),
  UNIQUE (EMAIL)
);
```

Tabela PRODUCER

```
CREATE TABLE PRODUCER(
  ID            INT          NOT NULL,
  PRODUCER_NAME VARCHAR(32)  NOT NULL,
  ADDRESS       VARCHAR(128) NOT NULL,
  PRIMARY KEY (ID),
  UNIQUE (PRODUCER_NAME)
);
```

Tabela PRODUCT

```
CREATE TABLE PRODUCT(
  ID            INT           NOT NULL,
  PRODUCT_CODE  VARCHAR(32)   NOT NULL,
  PRODUCT_NAME  VARCHAR(64)   NOT NULL,
  PRODUCT_PRICE NUMERIC(7, 2) NOT NULL,
  ADULTS_ONLY   BOOLEAN       NOT NULL,
  DESCRIPTION   TEXT          NOT NULL,
  PRODUCER_ID   INT           NOT NULL,
  PRIMARY KEY (ID),
  UNIQUE (PRODUCT_CODE),
  CONSTRAINT fk_product_producer
    FOREIGN KEY (PRODUCER_ID)
      REFERENCES PRODUCER (ID)
);
```

Tabela PURCHASE

```
CREATE TABLE PURCHASE(
  ID          INT  NOT NULL,
  CUSTOMER_ID INT  NOT NULL,
  PRODUCT_ID  INT  NOT NULL,
  QUANTITY    INT  NOT NULL,
  DATE_TIME   TIMESTAMP WITH TIME ZONE NOT NULL,
  PRIMARY KEY (ID),
  CONSTRAINT fk_purchase_customer
    FOREIGN KEY (CUSTOMER_ID)
      REFERENCES CUSTOMER (ID),
  CONSTRAINT fk_purchase_product
    FOREIGN KEY (PRODUCT_ID)
      REFERENCES PRODUCT (ID)
);
```

Tabela OPINION

```
CREATE TABLE OPINION(
  ID          INT  NOT NULL,
  CUSTOMER_ID INT  NOT NULL,
  PRODUCT_ID  INT  NOT NULL,
  STARS       INT  CHECK (STARS IN (1, 2, 3, 4, 5)) NOT NULL,
  COMMENT     TEXT  NOT NULL,
  DATE_TIME   TIMESTAMP WITH TIME ZONE NOT NULL,
  PRIMARY KEY (ID),
  CONSTRAINT fk_purchase_customer
    FOREIGN KEY (CUSTOMER_ID)
      REFERENCES CUSTOMER (ID),
  CONSTRAINT fk_purchase_product
    FOREIGN KEY (PRODUCT_ID)
      REFERENCES PRODUCT (ID)
);
```

executeUpdate()

Używana do zapytań INSERT, UPDATE, DELETE. Jest o tyle ciekawa, że jej nazwa zawiera słowo **update**, a

używa się jej również do operacji INSERT oraz DELETE. Wynikiem jej wywołania jest liczba zmodyfikowanych wierszy w bazie danych.

Przykład INSERT, UPDATE i DELETE:

Klasa JdbcConnectionExecuteUpdateExample

```
public class JdbcConnectionExecuteUpdateExample {

    public static void main(String[] args) {
        String datbaseURL = "jdbc:postgresql://localhost:5432/zajavka";
        String user = "postgres";
        String password = "password";
        try (
            Connection connection = DriverManager.getConnection(datbaseURL, user, password);
            Statement statement = connection.createStatement()
        ) {
            String query1 = "INSERT INTO PRODUCER (ID, PRODUCER_NAME, ADDRESS) " +
                "VALUES (21, 'Zajavka Group', 'Zajavkowa 15, Warszawa');";
            String query2 = "UPDATE PRODUCER SET ADDRESS = 'Nowy adres naszej siedziby' WHERE ID = 21;";
            String query3 = "DELETE FROM PRODUCER WHERE ID = 21;";

            Optional.of(statement.executeUpdate(query1))
                .ifPresent(result -> System.out.printf("Inserted %s row(s)%n", result));
            Optional.of(statement.executeUpdate(query2))
                .ifPresent(result -> System.out.printf("Updated %s row(s)%n", result));
            Optional.of(statement.executeUpdate(query3))
                .ifPresent(result -> System.out.printf("Deleted %s row(s)%n", result));
        } catch (Exception e) {
            System.err.printf("Error while working on database: %s%n", e.getMessage());
        }
    }
}
```

Każde wywołanie executeUpdate() zwraca ilość zmodyfikowanych wierszy.

Jeżeli natomiast chcielibyśmy wywołać komendę, która zwraca nam jakieś informacje, w tym celu możemy użyć executeQuery():

executeQuery()

Używana do zapytań `SELECT`, zwraca otrzymany z zapytania rezultat w postaci interfejsu `ResultSet`.

```
public class JdbcConnectionExecuteQueryExample {

    public static void main(String[] args) {
        String datbaseURL = "jdbc:postgresql://localhost:5432/zajavka";
        String user = "postgres";
        String password = "password";
        String query = "SELECT * FROM PRODUCER;";
        try (
            Connection connection = DriverManager.getConnection(datbaseURL, user, password);
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(query)
        ) {
            System.out.printf("Selected %s%n", resultSet);
        } catch (Exception e) {
            System.err.printf("Error while working on database: %s%n", e.getMessage());
        }

    }
}
```

Typem zwracanym z tego kodu jest `ResultSet` i sama próba wydrukowania go na ekranie kończy się czymś podobnym do wydruku poniżej - dołożone zostało słówko `Selected`.

```
Selected org.postgresql.jdbc.PgResultSet@4d49af10
```

Zaraz omówimy jak z takiego rezultatu `ResultSet` wyciągnąć faktyczne dane pobrane z bazy.

execute()

Dodajmy jeszcze natomiast, że oprócz `executeUpdate()` i `executeQuery()` istnieje sama metoda `execute()`. Można jej używać w następujący sposób:

Metoda `execute()` może być uruchomiona do zapytań `INSERT`, `UPDATE`, `DELETE` i `SELECT`. Jej wynikiem jest `boolean` mówiący, czy jako wynik został zwrócony `ResultSet`. Jest to o tyle ważne, że tylko zapytanie `SELECT` może zwrócić `ResultSet`. Reszta nie zwraca wyniku bo służy do modyfikacji stanu bazy danych.

```
public class JdbcConnectionExecuteExample {

    public static void main(String[] args) {
        String datbaseURL = "jdbc:postgresql://localhost:5432/zajavka";
        String user = "postgres";
        String password = "password";
        String query = "SELECT * FROM PRODUCER;";
        try (
            Connection connection = DriverManager.getConnection(datbaseURL, user, password);
            Statement statement = connection.createStatement()
        ) {
            boolean resultSetExists = statement.execute(query);
```

```
            if (resultSetExists) {
                ResultSet resultSet = statement.getResultSet();
                System.out.println("ResultSet: " + resultSet);
            } else {
                int count = statement.getUpdateCount();
                System.out.println("Count: " + count);
            }
        } catch (Exception e) {
            System.err.printf("Error while working on database: %s%n", e.getMessage());
        }
    }

}
```

Nie jest natomiast dobrym pomysłem zwracanie z metody raz `ResultSet` a raz `Integer`, więc potraktuj ten przykład z przymrużeniem oka ☺. Dalej będziemy się raczej skupiać na metodach `executeUpdate()` i `executeQuery()`. Oczywiście możemy próbować kombinować i wywołać `executeUpdate()` z SELECTem w środku, ale dostaniemy wtedy `SQLException`.

Podsumowując, do jakich rodzajów zapytań można stosować określone metody:

Metoda	INSERT	SELECT	UPDATE	DELETE
execute()	TAK	TAK	TAK	TAK
executeUpdate()	TAK	NIE	TAK	TAK
executeQuery()	NIE	TAK	NIE	NIE