PreparedStatement

Codice SQL più semplice con PreparedStatement

L'interfaccia **java.sql.PreparedStatement**, che estende **Statement**, permette di concepire più semplicemente e velocemente il codice SQL per l'interazione con il DB.

esempio input errato/malizioso

L'interfaccia **PreparedStatement** permette di scrivere **codice SQL parametrico**, sostituendo i dati in input concatenati via stringa con dei caratteri di punto interrogativo.

Fatto ciò, è possibile impostare uno ad uno i parametri espressi attraverso i metodi setter esposti da **PreparedStatement**.

Questi metodi permettono di inserire i parametri senza preoccuparsi della conflittualità dei loro contenuti: sarà JDBC, in collaborazione con il driver dello specifico DBMS, a risolvere ogni problema.

I setter di **PreparedStatement**, un po' come i getter di **ResultSet**, esistono per i principali tipi di Java.

Il seguente elenco li riassume:

- `setBoolean(int p, boolean value)` Imposta il booleano value come valore del parametro alla posizione p.
- `setByte(int p, byte value)` Imposta il byte value come valore del parametro alla posizione p.
- `setDate(int p, Date value)` Imposta l'oggetto java.util.Date value come valore del parametro alla posizione p.
- `setDouble(int p, double value)` Imposta il double value come valore del parametro alla posizione p.
- `setFloat(int p, float value)` Imposta il float value come valore del parametro alla posizione p.
- `setInt(int p, int value)` Imposta l'intero value come valore del parametro alla posizione p.
- `setLong(int p, long value)` Imposta il long value come valore del parametro alla posizione p.
- `setShort(int p, short value)` Imposta lo short value come valore del parametro alla posizione p.
- `setString(int p, String value)` Imposta la stringa value come valore del parametro alla posizione p.

Le **PreparedStatement**, oltre a risolvere i problemi illustrati, garantiscono inoltre maggiori prestazioni ed un alto grado di riusabilità. Quando si prepara un oggetto di questo tipo, infatti, il suo codice SQL viene precompilato.

La medesima **PreparedStatement**, inoltre, può essere sfruttata più volte consecutivamente, cambiando semplicemente i parametri attraverso i suoi metodi setter per ottenere risultati diversi, senza dover ogni volta riprocessare il codice SQL che le compone, come avverrebbe con una Statement classica.

Per lavorare con un oggetto **PreparedStatement** in luogo di un semplice Statement è necessario chiamare:

`PreparedStatement statement = connection.prepareStatement(CODICE\_SQL);` al posto di

`Statement statement = connection.createStatement();`

Il codice SQL, con **PreparedStatement**, va specificato al momento della creazione dell'oggetto, e non quando si richiamano i metodi executeQuery() o executeUpdate(). Le due varianti di questi metodi offerte da PreparedStatement, infatti, sono prive di argomenti.

L'esempio del paragrafo precedente può allora essere riscritto alla seguente maniera:

```
import java.io.*;
import java.sql.*;
public class JDBCTest5 {
// Nome del driver.
private static final String DRIVER = "com.mysql.jdbc.Driver";
// Indirizzo del database.
private static final String DB_URL =
"jdbc:mysql://localhost:3306/javatest";
// Questo metodo aggiunge un nuovo record alla tabella nel DB.
private static boolean aggiungiRecord(String nome, String cognome,
String indirizzo) {
// Preparo il riferimento alla connessione.
Connection connection = null;
try {
// Apro la connesione verso il database.
connection = DriverManager.getConnection(DB_URL);
// Preparo lo Statement per interagire con il database.
PreparedStatement statement = connection.prepareStatement(
"INSERT INTO Persone ( " +
" Nome, Cognome, Indirizzo " +
") VALUES ( " +
" ?, ?, ? " +
")"
);
// Imposto i parametri.
statement.setString(1, nome);
statement.setString(2, cognome);
statement.setString(3, indirizzo);
// Eseguo l'aggiornamento.
statement.executeUpdate();
return true;
} catch (SQLException e) {
// In caso di errore...
return false;
} finally {
if (connection != null) {
try {
connection.close();
} catch (Exception e) {
}
```

```
}
}
}
public static void main(String[] args) throws IOException {

// Interagisco con l'utente.
BufferedReader reader = new BufferedReader(
new InputStreamReader(System.in)
);
while (true) {
System.out.print("Nome: ");
String nome = reader.readLine();
System.out.print("Cognome: ");
String cognome = reader.readLine();
System.out.print("Indirizzo: ");
String indirizzo = reader.readLine();
System.out.println();
if (aggiungiRecord(nome, cognome, indirizzo)) {
System.out.println("Record aggiunto!");
} else {
System.out.println("Errore!");
}
System.out.println();
String ris;
do {
System.out.print("Vuoi aggiungerne un altro (si/no)? ");
ris = reader.readLine();
} while (!ris.equals("si") && !ris.equals("no"));
if (ris.equals("no")) {
break;
}
System.out.println();
}
}
}
```