JDBC Statement vs PreparedStatement – SQL Injection Example

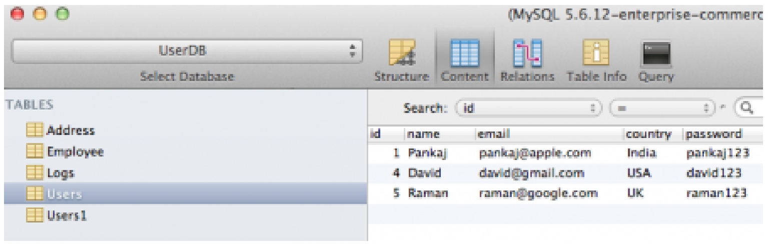
APRIL 2, 2018 BY PANKAJ — 14 COMMENTS

Today we will look into JDBC Statement vs PreparedStatement and some SQL Injection Example. While working with JDBC for database connectivity, we can use `Statement` or `PreparedStatement` to execute queries. These queries can be CRUD operation queries or even DDL queries to create or drop tables.

Statement vs PreparedStatement

Before comparing Statement vs PreparedStatement, let's see why we should avoid JDBC Statement. JDBC Statement has some major issues and should be avoided in all cases, let's see this with a simple example.

I have **Users** table in my local MySQL database with following data.



id	name	email	country	password
1	Pankaj	pankaj@apple.com	India	pankaj123
4	David	david@gmail.com	USA	david123
5	Raman	raman@google.com	UK	raman123

Below script will create the table and insert the data for test use.

```
CREATE TABLE `Users` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL DEFAULT '',
  `email` varchar(20) NOT NULL DEFAULT '',
  `country` varchar(20) DEFAULT 'USA',
  `password` varchar(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;

INSERT INTO `Users` (`id`, `name`, `email`, `country`, `password`)
VALUES
        (1, 'Pankaj', 'pankaj@apple.com', 'India', 'pankaj123'),
        (4, 'David', 'david@gmail.com', 'USA', 'david123'),
        (5, 'Raman', 'raman@google.com', 'UK', 'raman123');
```

A utility class for creating JDBC Connection to our mysql database.

DBConnection.java

```
package com.journaldev.jdbc.statements;

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

public class DBConnection {

        public final static String DB_DRIVER_CLASS = "com.mysql.jdbc.Driver";
        public final static String DB_URL = "jdbc:mysql://localhost:3306/UserDB";
```

```
	public final static String DB_USERNAME = "pankaj";
	public final static String DB_PASSWORD = "pankaj123";

	public static Connection getConnection() throws ClassNotFoundException,
SQLException {

		Connection con = null;

		// load the Driver Class
		Class.forName(DB_DRIVER_CLASS);

		// create the connection now
```

Now let's say we have following class that asks user to enter the email id and password and if it matches, then prints the user details. I am using JDBC Statement for executing the query.

`GetUserDetails.java`

```
package com.journaldev.jdbc.statements;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class GetUserDetails {

	public static void main(String[] args) throws ClassNotFoundException,
SQLException {

		//read user entered data
		Scanner scanner = new Scanner(System.in);
		System.out.println("Please enter email id:");
		String id = scanner.nextLine();
		System.out.println("User id="+id);
		System.out.println("Please enter password to get details:");
		String pwd = scanner.nextLine();
		System.out.println("User password="+pwd);
```

Let's see what happens when we pass different kinds of input to above program.

Valid User:

```
Please enter email id:
david@gmail.com
User id=david@gmail.com
Please enter password to get details:
david123
User password=david123
DB Connection created successfully
select name, country, password from Users where email = 'david@gmail.com' and
password='david123'
Name=David,country=USA,password=david123
```

So our program works fine and a valid user can enter their credentials and get his details.

Now let's see how a hacker can get unauthorized access to a user because we are using Statement for executing queries.

SQL Injection:

```
Please enter email id:
david@gmail.com' or '1'='1
User id=david@gmail.com' or '1'='1
Please enter password to get details:

User password=
DB Connection created successfully
select name, country, password from Users where email = 'david@gmail.com' or '1'='1'
and password=''
Name=David,country=USA,password=david123
```

As you can see that we are able to get the user details even without having password. The key point to note here is that query is created through String concatenation and if we provide proper input, we can hack the system, like here we did by passing user id as `david@gmail.com' or '1'='1`.

This is an example of **SQL Injection** where poor programming is responsible for making our application vulnerable for unauthorized database access.

One solution is to read the user input and then escape all the special characters that are used by MySQL but that would be clumsy and error prone. That's why JDBC API came up with `PreparedStatement` interface that extends `Statement` and automatically escape the special characters before executing the query.

Let's rewrite above class using PreparedStatement and try to hack the system.

GetUserDetailsUsingPS.java

```
package com.journaldev.jdbc.statements;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

public class GetUserDetailsUsingPS {

        public static void main(String[] args) throws ClassNotFoundException,
SQLException {

                // read user entered data
                Scanner scanner = new Scanner(System.in);
                System.out.println("Please enter email id:");
                String id = scanner.nextLine();
                System.out.println("User id=" + id);
                System.out.println("Please enter password to get details:");
                String pwd = scanner.nextLine();
                System.out.println("User password=" + pwd);
                printUserData(id, pwd);
```

Now if we will try to hack the system, let's see what happens.

SQL Injection:

```
Please enter email id:
david@gmail.com' or '1'='1
User id=david@gmail.com' or '1'='1
Please enter password to get details:

User password=
DB Connection created successfully
```

So we are not able to hack the database, it happened because the actual query that is getting executed is:

```
select name, country, password from Users where email = 'david@gmail.com\' or \'1\'=\'1\'
and password=''
```

When we fire a query to be executed for a relational database, it goes through following steps.

1. Parsing of SQL query
2. Compilation of SQL Query
3. Planning and optimization of data acquisition path
4. Executing the optimized query and return the resulted data

When we use `Statement`, it goes through all the four steps but with `PreparedStatement` first three steps are executed when we create the prepared statement. So execution of query takes less time and more quick that Statement.

Another benefit of using PreparedStatement is that we can use Batch Processing through `addBatch()` and `executeBatch()` methods. We can create a single prepared statement and use it to execute multiple queries.

Some points to remember about JDBC PreparedStatement are:

1. PreparedStatement helps us in preventing SQL injection attacks because it automatically escapes the special characters.
2. PreparedStatement allows us to execute dynamic queries with parameter inputs.
3. PreparedStatement provides different types of setter methods to set the input parameters for the query.
4. PreparedStatement is faster than Statement. It becomes more visible when we reuse the PreparedStatement or use it's batch processing methods for executing multiple queries.
5. PreparedStatement helps us in writing object Oriented code with setter methods whereas with Statement we have to use String Concatenation to create the query. If there are multiple parameters to set, writing Query using String concatenation looks very ugly and error prone.
6. PreparedStatement returns `FORWARD_ONLY` ResultSet, so we can only move in forward direction.
7. Unlike Java Arrays or List, the indexing of PreparedStatement variables starts with 1.
8. One of the limitation of PreparedStatement is that we can't use it for SQL queries with IN clause because PreparedStatement doesn't allow us to bind multiple values for single placeholder (?). However there are few alternative approaches to use PreparedStatement for IN clause, read more at JDBC PreparedStatement IN clause.

That's all for the comparison of JDBC Statement vs PreparedStatement. You should always use PreparedStatement because it's fast, object oriented, dynamic and more reliable.

« PREVIOUS
Java JDBC Transaction Management and Savepoint

NEXT »
JDBC Batch insert update MySQL Oracle

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DATABASE, JAVA

Comments

saima khan says
AUGUST 17, 2017 AT 8:19 AM

Thank You So much. It was helpful.
but could you please explain how a hacker can hack the sql code (sql injection).
Reply

Amit says
OCTOBER 12, 2016 AT 6:55 PM
great explanation !
Reply

JHON says
MAY 7, 2016 AT 7:46 AM
IT GOOD FOR POOR PEOPLES LIKE ME
Reply

fleetwu says
FEBRUARY 6, 2016 AT 8:47 AM
It is really helpful. Thank you.
Reply

Sandeep Kumar says
JANUARY 25, 2016 AT 5:02 AM
Well done….
Reply

n s says
OCTOBER 1, 2015 AT 1:38 AM
Thank you for this write-up
Reply

sheikh says
JULY 7, 2015 AT 1:05 PM

Plz stop to make people fool
u don't have idea what r u taking about
The query will only fired succesfully if it have regular syntax and without matching
of parameters it is not posible to get success for hackers also
DBMS's are highly secured databases not like spreadsheeets
Reply

Danish says
OCTOBER 1, 2015 AT 10:22 PM
Even in the case of statement, password is not showing. He made us fool.
Reply

Pankaj says
OCTOBER 2, 2015 AT 5:56 AM
Hmm, I am sure you guys have not heard of brute force attacks, DDoS etc. Sometimes "Ignorance is Bliss".
Reply

Yadav says
APRIL 2, 2015 AT 11:40 AM
Thanks!
Very helpful and well described tutorial
Reply

ammu says
NOVEMBER 14, 2014 AT 4:02 PM
Hi Pankaj,
You didn't explain that how hacker can hack the code. Whatever you wrote that is not well explained.
Reply

Dave says
JUNE 5, 2014 AT 8:01 AM
This was helpful, thanks!

Reply

Shashank says

APRIL 30, 2014 AT 3:08 AM

In 2nd query select name, country, password from Users where email = 'david@gmail.com' or '1'='1' and password="" , password is empty ans we are doing and operation , how it will return row as out put ?

Reply

David says

MAY 16, 2014 AT 9:03 AM

This does look wrong. I think password should also have some sql injection e.g.
enter password
pwd' or '1'='1
because as you point out, the example in the blog post will only print rows where password=""

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

CORE JAVA TUTORIAL

Java 10 Tutorials Java 9 Tutorials Java 8 Tutorials Java 7 Tutorials Core Java Basics OOPS Concepts Data Types and Operators String Manipulation Java Arrays Annotation and Enum Java Collections Java IO Operations Java Exception Handling MultiThreading and Concurrency Regular Expressions Advanced Java Concepts

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial

SQL Injection Tutorial

Web Application Security Tool

What is SQL Injection

Web Application Firewall

SQL Server Injection

Learn Programming Online

Learn Java in 7 Days

Create Your Own Website