Prepared Statements

Denne artikel beskriver hvorfor prepared statements er gode.

Den forudsætter lidt kendskab til Java og JDBC.

Skrevet den **18. Feb 2010** af **arne_v** I kategorien **Programmering / Java** | ★★★☆☆

Historie:
V1.0 - 12/11/2005 - original
V1.1 - 14/11/2005 - fix manglende ' i eksempel
V1.2 - 18/02/2010 - smårettelser

Problemerne

Alle der har programmeret op mod en database kender et eller flere af problemerne.

1) uheldige single quotes

Eksempel:

```
sql = "INSERT INTO tt VALUES(" + id + ",'" + name + "')";
```

hvis navnet er Hansen så virker det fint:

```
INSERT INTO tt VALUES(123,'Hansen')
```

men hvis navnet er O'Toole så giver det fejl:

```
INSERT INTO tt VALUES(123,'O'Toole')
```

2) single quotes med vilje (kendt som SQL injection)

Eksempel:

```
sql = "SELECT * FROM myusers WHERE un = '" + username + "' AND pw = '" + password + "'";
```

(efterfulgt af et test på om der blev fundet nogle records)

det virker fint for den pæne bruger som indtaster:
arne
hemmeligt

```
SELECT * FROM myusers WHERE un = 'arne' AND pw = 'hemmeligt'
```

men det returnerer forkert OK for den ondsinded cracker som indtaster:
arne
x' OR 'x' = 'x

SELECT * FROM myusers WHERE un = 'arne' AND pw = 'x' OR 'x' = 'x'

3) dato formater

Den giver altid problemer med input til databasen.

Hvilket format skal man bruge:
  dd mm yyyy (dansk)
  mm dd yyyy (US)
  yyyy mm dd (sorterings rigtigt)
?

Skal værdierne:
  sættes i ''
  sættes i ##
  konverteres med en funktion
?

Styres formatet af:
  operativ system sprog version
  database sprog version
  styre system sprog indstilling
  database sprog indstilling
?

Løsningen

En begynder løsning på de 2 første problemer er at fordoble
alle single quotes:

sql = "INSERT INTO tt VALUES(" + id + ",'" + name.replaceAll("'","''") + "')";

INSERT INTO tt VALUES(123,'Hansen')

INSERT INTO tt VALUES(123,'O''Toole')

sql = "SELECT * FROM myusers WHERE un = '" + username.replaceAll("'","''") + "' AND pw = '" +
password.replaceAll("'","''") + "'";

SELECT * FROM myusers WHERE un = 'arne' AND pw = 'hemmeligt'

SELECT * FROM myusers WHERE un = 'arne' AND pw = 'x'' OR ''x'' = ''x'

og det virker, men det er ikke super godt:
  * det er ikke kønt
  * det er nemt at glemme
  * forskellige databaser kan have forskellige andre tegn som også kan misbruges
  * det løser ikke dato problemet

Dato problemet undlader man ofte helt at løse. Man hardkoder SQL sætningerne
med formatet til det system man udvikler på. Enten med en SimpleDateFormat eller
ved noget banal string manipulation.

Og så får man problemet når man skal have det til at køre på en anden
maskine.

Den rigtige løsning som løser alle problemerne er at bruge PreparedStatement fremfor almindelig Statement.

Med en PreparedStatement skriver man bare ? alle de steder i ens SQL hvor der skal indsættes værdier og så set'er man de værdier og set metoden håndterer alle problemerne.

Det lyder måske lidt mystisk, men lad os tage nogle eksempler.

Kode eksempler

Eksemplerne vil bruge Oracle som database, men alle eksemplerne virker lige så godt med SQLServer eller MySQL, man skal bare rette driver navn og connection URL.

De data som køres på er:

```
CREATE TABLE tt (
    id INTEGER PRIMARY KEY,
    name VARCHAR(50)
);

CREATE TABLE myusers (
    un VARCHAR(32) PRIMARY KEY,
    pw VARCHAR(32)
);

INSERT INTO myusers VALUES('arne', 'hemmeligt');

CREATE TABLE dtest (
    i INTEGER PRIMARY KEY,
    d DATE
);
```

Først INSERT med single quotes:

TestPrep1.java:

```
package testprep;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class TestPrep1 {
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.OracleDriver");
        Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:arnepc3",
"arne", "xxxx");
```

```
        PreparedStatement pstmt = con.prepareStatement("INSERT INTO tt VALUES
(?, ?)");
        pstmt.setInt(1, 123);
        pstmt.setString(2, "Hansen");
        pstmt.executeUpdate();
        pstmt.setInt(1, 124);
        pstmt.setString(2, "O'Toole");
        pstmt.executeUpdate();
        pstmt.close();
        con.close();
    }
}
```

Bemærk at vi ikke sætter '' omkring ? når det er en String.

Og en gang mere.

TestPrep2.java:

```
package testprep;

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

public class TestPrep2 {
    public static boolean isValid(String un, String pw) throws SQLException {
        Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:arnepc3",
"arne", "xxxx");
        PreparedStatement pstmt = con.prepareStatement("SELECT * FROM myusers
WHERE un = ? AND pw = ?");
        pstmt.setString(1, un);
        pstmt.setString(2, pw);
        ResultSet rs = pstmt.executeQuery();
        boolean res = rs.next();
        rs.close();
        pstmt.close();
        con.close();
        return res;
    }
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.OracleDriver");
        System.out.println(isValid("anonymous", ""));
        System.out.println(isValid("arne", "hemmeligt"));
        System.out.println(isValid("arne", "x' OR 'x' = 'x"));
    }
}
```

Og til sidst dato.

TestPrep3.java:

```
package testprep;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Timestamp;
import java.util.Calendar;

public class TestPrep3 {
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.OracleDriver");
        Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:arnepc3",
"arne", "xxxx");
        PreparedStatement ins = con.prepareStatement("INSERT INTO dtest VALUES
(?, ?)");
        for(int i = 0; i < 10; i++) {
            ins.setInt(1, i);
            Timestamp ts = new
Timestamp(Calendar.getInstance().getTimeInMillis());
            ins.setTimestamp(2, ts);
            ins.executeUpdate();
            Thread.sleep(1000);
        }
        ins.close();
        PreparedStatement sel = con.prepareStatement("SELECT * FROM dtest
WHERE d > ?");
        Timestamp cut = new Timestamp(Calendar.getInstance().getTimeInMillis()
- 5000);
        sel.setTimestamp(1, cut);
        ResultSet rs = sel.executeQuery();
        while(rs.next()) {
            int i = rs.getInt(1);
            Timestamp ts = rs.getTimestamp(2);
            System.out.println(i + " " + ts);
        }
        rs.close();
        sel.close();
        con.close();
    }
}
```

Performance

Effekten på performance af at bruge PreparedStatement afhænger af driveren.

For nogle drivere er der ikke forskel på PreparedStatement og Statement.

For andre drivere er det dyrt at lave en PreparedStatement men hurtigere at executeUpdate.

Her er en lille test.

TestPrep4.java

```
package testprep;

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

public class TestPrep4 {
    private static final int N = 10000;
    private static void delete(Connection con) throws SQLException {
        Statement stmt = con.createStatement();
        stmt.executeUpdate("DELETE FROM tt");
        stmt.close();
    }
    private static void testStatSing(Connection con) throws SQLException {
        delete(con);
        long t1 = System.currentTimeMillis();
        for(int i = 0; i < N; i++) {
            Statement stmt = con.createStatement();
            String s = "Noget tekst bla. bla. bla.";
            stmt.executeUpdate("INSERT INTO tt VALUES (" + i + ",'" + s +
"')");
            stmt.close();
        }
        long t2 = System.currentTimeMillis();
        System.out.println("Statement ikke genbrugt: " + (t2 - t1));
    }
    private static void testStatMulti(Connection con) throws SQLException {
        delete(con);
        long t1 = System.currentTimeMillis();
        Statement stmt = con.createStatement();
        for(int i = 0; i < N; i++) {
            String s = "Noget tekst bla. bla. bla.";
            stmt.executeUpdate("INSERT INTO tt VALUES (" + i + ",'" + s +
"')");
        }
        stmt.close();
        long t2 = System.currentTimeMillis();
        System.out.println("Statement genbrugt: " + (t2 - t1));
    }
    private static void testPrepStatSing(Connection con) throws SQLException {
        delete(con);
        long t1 = System.currentTimeMillis();
```

```
        for(int i = 0; i < N; i++) {
            PreparedStatement pstmt = con.prepareStatement("INSERT INTO tt
VALUES (?, ?)");
            pstmt.setInt(1, i);
            pstmt.setString(2, "Noget tekst bla. bla. bla.");
            pstmt.executeUpdate();
            pstmt.close();
        }
        long t2 = System.currentTimeMillis();
        System.out.println("PreparedStatement ikke genbrugt: " + (t2 - t1));
    }
    private static void testPrepStatMulti(Connection con) throws SQLException
{
        delete(con);
        long t1 = System.currentTimeMillis();
        PreparedStatement pstmt = con.prepareStatement("INSERT INTO tt VALUES
(?, ?)");
        for(int i = 0; i < N; i++) {
            pstmt.setInt(1, i);
            pstmt.setString(2, "Noget tekst bla. bla. bla.");
            pstmt.executeUpdate();
        }
        pstmt.close();
        long t2 = System.currentTimeMillis();
        System.out.println("PreparedStatement genbrugt: " + (t2 - t1));
    }
    private static void test(String txt, String url, String un, String pw)
throws SQLException {
        System.out.println(txt);
        Connection con = DriverManager.getConnection(url, un, pw);
        testStatSing(con);
        testStatMulti(con);
        testPrepStatSing(con);
        testPrepStatMulti(con);
        con.close();
    }
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.OracleDriver");
        Class.forName("com.mysql.jdbc.Driver");
        test("Oracle", "jdbc:oracle:thin:@localhost:1521:arnepc3", "arne",
"xxxx");
        test("MySQL", "jdbc:mysql://localhost/Test", "", "");
    }
}
```

Resultat:

```
Oracle
Statement ikke genbrugt: 11782
Statement genbrugt: 9516
PreparedStatement ikke genbrugt: 8266
PreparedStatement genbrugt: 4844
```

```
MySQL
Statement ikke genbrugt: 1500
Statement genbrugt: 1468
PreparedStatement ikke genbrugt: 4032
PreparedStatement genbrugt: 2453
```

I skal ikke kigge på de absolutte tal (MySQL bruger MyISAM tabeller uden transaktion log).

Men det fremgår at for Oracle er PreparedStatement hurtigere end Statement, mens for MySQL er PreparedStatement langsommere end Statement.

Formentligt vil MySQL vise bedre resultater for PreparedStatement med en nyere MySQL server og en nyere MySQL JDBC driver, som understøtter server side prepared statement.

Konklusion

Det er en selvfølge at man bruger PreparedStatement når man skal igang med seriøs brug af JDBC.

Eksemplerne i denne artikel skulle gerne have givet et indblik i hvordan man bruger PreparedStatement.

Kommentar af jensgram d. 13. Nov 2005 | 1

Nydeligt, men mener du ikke x' OR 'x' = 'x i stedet for x OR 'x' = 'x? Eller er det mig, der er helt galt på den. /Jens Gram

Kommentar af general_custer d. 27. May 2006 | 2

Kommentar af thesurfer d. 27. May 2007 | 3

God artikel om sikkerheden omkring indsættelse af data. Artiklen behøvede dog ikke at koncentrere sig om JDBC og Java, men kunne være skrevet lidt mere generelt, da problemet eksisterer i (sådan set) samtlige databaser der findes. Angående det omtalte hack, er det nok med bare at skrive: ' or '' = '

Kommentar af heine112 d. 31. Jan 2006 | 4