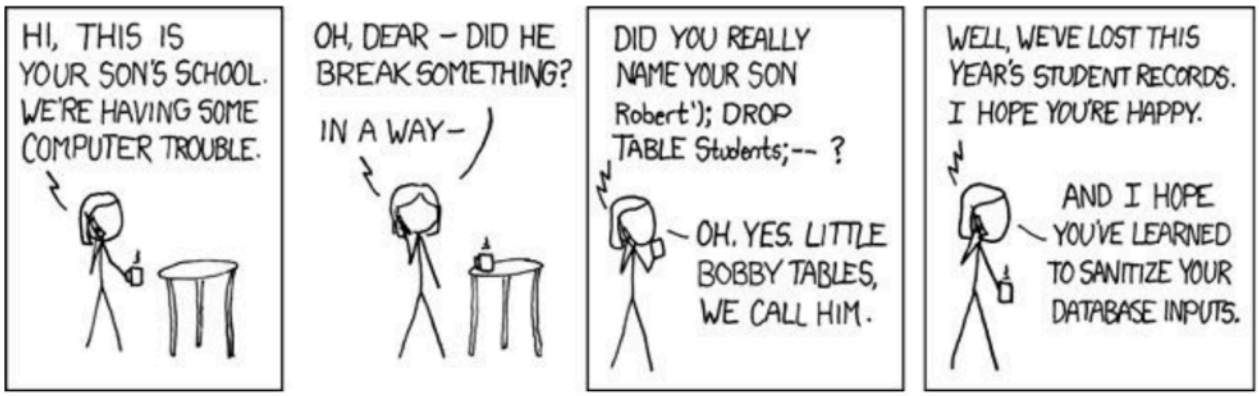
Documento a cura di Simone Remoli.

PREPARED STATEMENTS.

Usa sempre Prepared Statements per proteggere il database dagli attacchi **SQL Injection!**

Vediamo alcune problematiche :)

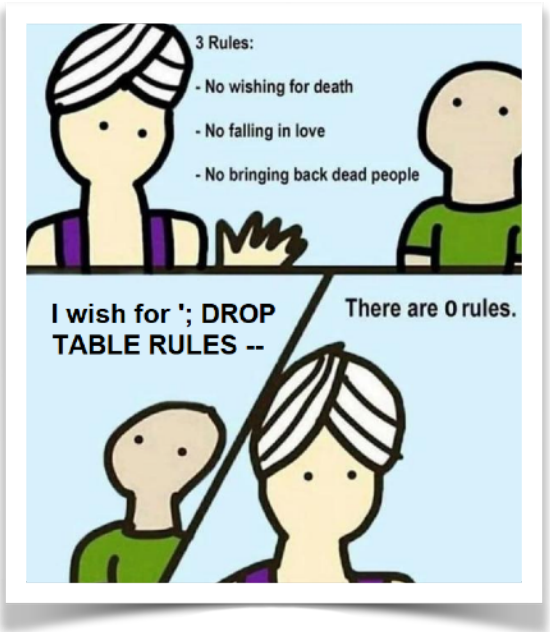


Una scuola chiama la madre di uno studente per segnalare un problema con il loro sistema informatico.
La madre chiede: **"Ha rotto qualcosa?"**, e l'operatore risponde **"In un certo senso…"**. Il problema è che il figlio si chiama "**Robert'); DROP TABLE Students;–** ".
Il responsabile IT della scuola, frustrato, dice: **"Abbiamo perso tutti i dati degli studenti di quest'anno. Spero che tu sia felice."**
La madre aggiunge poi: "E spero che abbiate imparato a sanificare gli input del database".

Che cosa è successo?
Il nome viene passato direttamente in una query SQL senza validazione, causando l'esecuzione della seguente istruzione SQL:

```
INSERT INTO studenti (nome) VALUES ('Robert'); DROP TABLE Students;--');
```

Il comando **DROP TABLE Students** cancella l'intera tabella degli studenti.



Il genio esprime tre regole:

- Non si può desiderare la morte di qualcuno.
- Non si può far innamorare qualcuno.
- Non si possono riportare in vita i morti.
-

L'utente formula il suo desiderio:

```
'; DROP TABLE RULES --
```

Qui l'utente **inietta un comando SQL dannoso** invece di esprimere un desiderio normale.
Il genio esegue il desiderio e la tabella RULES, che conteneva le tre regole, viene **cancellata** dal database.

Di cosa stiamo parlando?

Le vignette fanno riferimento a **SQL Injection**, un attacco in cui l'input utente non viene validato e viene interpretato direttamente come codice SQL.

Come si previene SQL Injection?

Utilizzare i Prepared Statements, che separano il codice SQL dai dati dell'utente.

Esempio vulnerabile a SQL Injection:

```
String query = "SELECT * FROM utenti WHERE username = '" + inputUtente + "'";
Statement stmt = conn.createStatement();
stmt.executeUpdate(query);
```

Se l'utente inserisce ' OR 1=1 --, la query diventa:

```
SELECT * FROM utenti WHERE username = '' OR 1=1 –';
```

Esempio sicuro con Prepared Statements:

```
String query = "SELECT * FROM utenti WHERE username = ?";
PreparedStatement pstmt = conn.prepareStatement(query);
pstmt.setString(1, inputUtente);
pstmt.executeQuery();
```

Qui l'input viene trattato come valore e non come codice SQL eseguibile!

I **prepared statement** sono una tecnica che consente di inviare al DBMS una "**query incompleta**", ossia con dei placeholder:

- **INSERT INTO score (event,id,student,id,score) VALUES(?,?,?)**

Successivamente, si possono "collegare" a tale query dei parametri.
Quando il DBMS riceve questi parametri, esegue la query completa e restituisce i risultati.

PreparedStatement è una classe figlia di Statement.

ESEMPIO DI LETTURA.

```
PreparedStatement stmt = con.prepareStatement(
          "select Stipendio from Impiegato where Cognome = ?");
stmt.setString(1, "Bianchi");
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
      int stipendio = rs.getInt(1);
      System.out.println("Stipendio = " + stipendio);
}
rs.close();
stmt.close();
```

ESEMPIO DI SCRITTURA.

```
String updateString =
      "update Impiegato set Stipendio = ? where Cognome = ?";
PreparedStatement updateSalary =
      con.prepareStatement(updateString);
updateSalary.setInt(1, 2500);
updateSalary.setString(2, "Bianchi");
int n = updateSalary.executeUpdate(); // n: Affected rows
updateSalary.close();
```

ESEMPI COMPLETI.

```
String query = "SELECT * FROM utenti WHERE username = ?";
PreparedStatement pstmt = conn.prepareStatement(query);
pstmt.setString(1, "mario");
ResultSet rs = pstmt.executeQuery();

while (rs.next())
{
    System.out.println("Nome: " + rs.getString("nome"));
}
```

```java
String query = "UPDATE utenti SET email = ? WHERE username = ?”;
PreparedStatement pstmt = conn.prepareStatement(query);
pstmt.setString(1, "nuovaemail@example.com");
pstmt.setString(2, "mario");
pstmt.executeUpdate();
```

```java
String query = "DELETE FROM utenti WHERE username = ?";
PreparedStatement pstmt = conn.prepareStatement(query);
pstmt.setString(1, “mario");
pstmt.executeUpdate();
```

Chiamare una Stored Procedure con CallableStatement.

```java
CallableStatement cstmt = conn.prepareCall(“{CALL getUserEmail(?, ?)}");
// Imposta il parametro di input
cstmt.setString(1, "mario");
// Registra il parametro di output
cstmt.registerOutParameter(2, Types.VARCHAR);
// Esegue la stored procedure
cstmt.execute();
// Ottiene il valore di output
String email = cstmt.getString(2);
System.out.println("Email: " + email);
```

E supponi di avere una store procedure così:

```sql
CREATE PROCEDURE getUserEmail(IN userName VARCHAR(50), OUT userEmail VARCHAR(100))
BEGIN
    SELECT email INTO userEmail FROM utenti WHERE username = userName;
END;
```

Il codice java esegue la stored procedure.
Recupera e stampa il valore restituito.

Ma se una stored procedure restituisce più risultati?
Supponiamo di avere il seguente codice java:

```java
import java.sql.*;

public class CallableStatementExample {
    public static void main(String[] args)
      {
        String url = "jdbc:mysql://localhost:3306/tuo_database";
        String user = "root";
        String password = "password";

        try (Connection conn = DriverManager.getConnection(url, user, password))
        {

            // Prepara la chiamata alla stored procedure
            CallableStatement cstmt = conn.prepareCall("{CALL getUserDetails(?)}");

            // Imposta il parametro di input
            cstmt.setString(1, "mario");

            // Esegue la procedura e ottiene i risultati
            ResultSet rs = cstmt.executeQuery();

            // Itera sui risultati e li stampa
            while (rs.next())
            {
                String nome = rs.getString("nome");
                String cognome = rs.getString("cognome");
                String email = rs.getString("email");

                System.out.println("Nome: " + nome);
                System.out.println("Cognome: " + cognome);
                System.out.println("Email: " + email);
            }

          } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

E la seguente SP:

```sql
CREATE PROCEDURE getUserDetails(IN userName VARCHAR(50))
BEGIN
    SELECT nome, cognome, email FROM utenti WHERE username = userName;
END;
```

Effettivamente il codice restituisce una proiezione di nome, cognome e email di Mario.

In generale:

```
CallableStatement cs = con.prepareCall("{call proc(?,?,?)}");
cs.setString(1, "argument");
cs.setFloat(2, 1.);
cs.registerOutParameter(3, Types.NUMERIC);
cs.setFloat(3, 123.45);
ResultSet rs = cs.executeQuery();

while (rs.next()) {
    ...
}
float f = cs.getFloat(3);
```

I parametri in output vanno registrati con `registerOutParameter()`.

I parametri INOUT devono sia ricevere un valore che essere registrati.