Guia de Codificação Segura: Proteção contra Injeção de SQL/NoSQL

Injeção de código (SQL/NoSQL) é uma vulnerabilidade crítica em aplicações de dados. Trata-se de situações em que entradas maliciosas do usuário são inseridas em consultas de banco, alterando sua lógica e permitindo acesso indevido. Segundo a MDN, a injeção de SQL "pode ganhar acesso não autorizado ao banco de dados" e está na raiz de muitos vazamentos de dados. Em outras palavras, é uma das falhas mais graves que um aplicativo que acessa banco de dados pode sofrer. Por exemplo, se o usuário inserir no campo de nome a string `"' OR '1'='1"`, um código vulnerável pode gerar a consulta:

```
SELECT * FROM users WHERE name = '' OR '1'='1'
```

Como `'1'='1'` é sempre verdadeiro, a cláusula inteira passa a ser verdadeira e retorna todas as linhas da tabela, permitindo ao invasor burlar a autenticação.

O problema decorre de consultas construídas dinamicamente via concatenação de strings contendo dados do usuário. Na prática, qualquer query que "use concatenação de strings e entrada do usuário" fica suscetível a injeção. Portanto, o primeiro princípio é: não monte consultas SQL concatenando diretamente dados vindos do usuário.

Exemplo de Código Vulnerável (Java/JDBC)

Considere este código Java genérico de login, que concatena diretamente a entrada do usuário na query SQL:

```
String userInput = request.getParameter("username");
String sql = "SELECT * FROM users WHERE name = '" + userInput + "'";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

Aqui o valor de `userInput` é inserido literalmente na string SQL sem nenhuma validação nem separação de dados. Se `userInput` for, por exemplo, `" OR '1'='1"`, a consulta resultante será:

```
SELECT * FROM users WHERE name = '' OR '1'='1'
```

Neste caso, a cláusula `OR '1'='1'` faz com que a condição seja sempre verdadeira, retornando todas as linhas existentes. Assim, basta um usuário mal-intencionado inserir esse fragmento para burlar completamente a autenticação. Quando se permite *string concatenation*, "um invasor pode inserir código SQL e a aplicação executará o código do invasor no banco de dados". Em resumo: ao concatenar dados não tratados na query, não há distinção entre código SQL e informação do usuário, abrindo brechas para injeção.

Prevenção com PreparedStatement (JDBC)

A solução recomendada é usar PreparedStatement (consultas parametrizadas) em vez de Statement. Nesse modelo, a query é definida estática com placeholders (?), e os valores são associados separadamente, assim:

```
String userInput = request.getParameter("username");
String sql = "SELECT * FROM users WHERE name = ?";
PreparedStatement pstmt = connection.prepareStatement(sql);
pstmt.setString(1, userInput);
ResultSet rs = pstmt.executeQuery();
```

Dessa forma, o valor de `userInput` não é concatenado no SQL, mas passado como **dado literal**. O banco de dados **sempre distinguirá código de dados**, independentemente do conteúdo fornecido. Em outras palavras, mesmo que o usuário insira `"' OR '1'='1"`, o mecanismo de parâmetros fará a consulta pelo nome exato `"' OR '1'='1"`, sem executar o `OR`. Com *prepared statements*, "um invasor não pode alterar o propósito de uma consulta, mesmo inserindo comandos SQL". Se um atacante informar `tom' OR '1'='1` como entrada, a consulta parametrizada procurará **literalmente** esse nome completo, deixando o banco protegido contra códigos maliciosos. Ou seja, a consulta só retorna resultados se houver um usuário cujo nome inclua exatamente aquele texto com aspas.

Prevenção com ORM (Hibernate/JPA)

Frameworks ORM (por exemplo, Hibernate/JPA) internamente já suportam consultas parametrizadas. Mesmo em HQL (Hibernate Query Language) ou JPQL, deve-se evitar concatenar strings com valores de usuário. Exemplo de abordagem **vulnerável** em HQL:

```
Query unsafeQuery = session.createQuery(
```

```
    "FROM User WHERE name = '" + userInput + "'"
);
```

Em vez disso, use **parâmetros nomeados**:

```
Query safeQuery = session.createQuery(
    "FROM User WHERE name = :username"
);
safeQuery.setParameter("username", userInput);
```

Assim, o Hibernate trata `userInput` como dado. Como no caso do PreparedStatement, primeiro define-se a consulta fixa (com `:username`) e depois vincula-se o valor do usuário. O OWASP observa que todos os ORMs modernos suportam essa abordagem parametrizada, eliminando a possibilidade de injetar código malicioso através dos valores.

Outras Medidas de Prevenção

Além de usar consultas parametrizadas, recomenda-se adotar boas práticas adicionais:

- **Validação de Entrada (Whitelist)**: Sempre que possível, filtre e restrinja o formato dos dados antes de usá-los. Por exemplo, use expressões regulares para permitir somente letras em campos de nome, converte strings numéricas com `Integer.parseInt`, etc. Em cenários em que parte da query (nome de tabela/coluna) não aceita `?`, use *listas permitidas* pré-definidas. A validação serve como segunda linha de defesa.

- **Princípio do Menor Privilégio**: Configure o usuário do banco de dados usado pela aplicação para ter **apenas as permissões necessárias**. Por exemplo, um usuário de conexão que só precise fazer `SELECT` (ou `UPDATE/INSERT` em tabelas específicas) reduz o impacto de uma injeção bem-sucedida. Recomendo evitar usar a conta de administrador: cada aplicação deve ter seu próprio usuário de BD com acessos restritos.

- **Evitar Escape Manual de Strings**: Embora existam funções como `mysql_real_escape_string`, elas são menos confiáveis que a parametrização. A estratégia de **escapar toda entrada de usuário** é "fortemente desencorajada" ("strongly discouraged"). Isso porque escapes podem falhar dependendo do banco ou codificação. Em vez de tentar

proteger manualmente as strings, é mais seguro deixar o driver parametrizado cuidar disso.

- **Stored Procedures Seguras**: Procedimentos armazenados (stored procedures) também podem prevenir injeção *se forem escritos corretamente*. Elas só são tão seguras quanto o cuidado dado internamente: uma procedure que concatena strings dos parâmetros estará vulnerável, mas se ela receber parâmetros (como `CALL myProc(?)`), seu efeito é análogo a um PreparedStatement. Quando bem escritas (com parâmetros), stored procedures oferecem segurança similar às consultas parametrizadas.

Em resumo, a ideia central é **tratar dados como dados**, nunca como parte do código SQL. Manter a query fixa e enviar apenas valores parametrizados fecha a porta para que um invasor "brinque" com a lógica do SQL.

Injeção em Bancos NoSQL

Sistemas NoSQL (MongoDB, CouchDB, etc.) são suscetíveis a ataques similares quando aceitam entradas arbitrárias na consulta. Como o Imperva alerta, a maior parte das vulnerabilidades de injeção NoSQL ocorre porque desenvolvedores aceitam e processam entradas de usuários **sem sanitização adequada**. Por exemplo, em MongoDB é arriscado montar consultas JSON como string concatenada. Em vez disso, use sempre **APIs nativas do driver** que constroem queries a partir de objetos (por exemplo, `collection.find({ name: userInput })`), garantindo que o conteúdo é tratado como valor. Além disso, é essencial validar/filtrar os campos esperados, evitando que o usuário injete operadores ou código inesperado. Especial atenção: não use funções que executem código arbitrário (como o operador `$where` em MongoDB com código JavaScript inserido), pois elas podem ser exploradas. Em suma, o princípio "dados como dados" vale também para NoSQL: nunca coloque diretamente em uma consulta strings vindas de usuário. Sempre construa suas consultas via parâmetros ou objetos, e valide o conteúdo enviado.

Comparação: Vulnerável × Seguro

- **Concatenação de strings (Vulnerável)** – Exemplo JDBC:

```
String sql = "SELECT * FROM users WHERE name = '" + userInput + "'";
```

Esse padrão permite injetar fragmentos como `'' OR '1'='1"` na consulta, mudando a lógica do SQL. Na prática, basta um único input malicioso para comprometer a query.

Consulta Parametrizada (Seguro) – Usando PreparedStatement (JDBC) ou parâmetros (ORM):

```
// JDBC
String sql = "SELECT * FROM users WHERE name = ?";
PreparedStatement pstmt = connection.prepareStatement(sql);
pstmt.setString(1, userInput);
```

Aqui `userInput` é tratado como dado sem alterar o SQL. A query resultante buscará literalmente o valor informado, sem executar nenhum código injetado. Como explica o OWASP, o banco "sempre distinguirá código de dados" com esse estilo de consulta.

ORM (Hibernate/JPA) (Seguro) – Usar parâmetros nomeados:

```
Query q = session.createQuery("FROM User WHERE name = :username");
q.setParameter("username", userInput);
```

Novamente, `:username` separa código de dado. Todos os ORMs modernos suportam essa abordagem e eliminam o risco de injeção nessa camada.

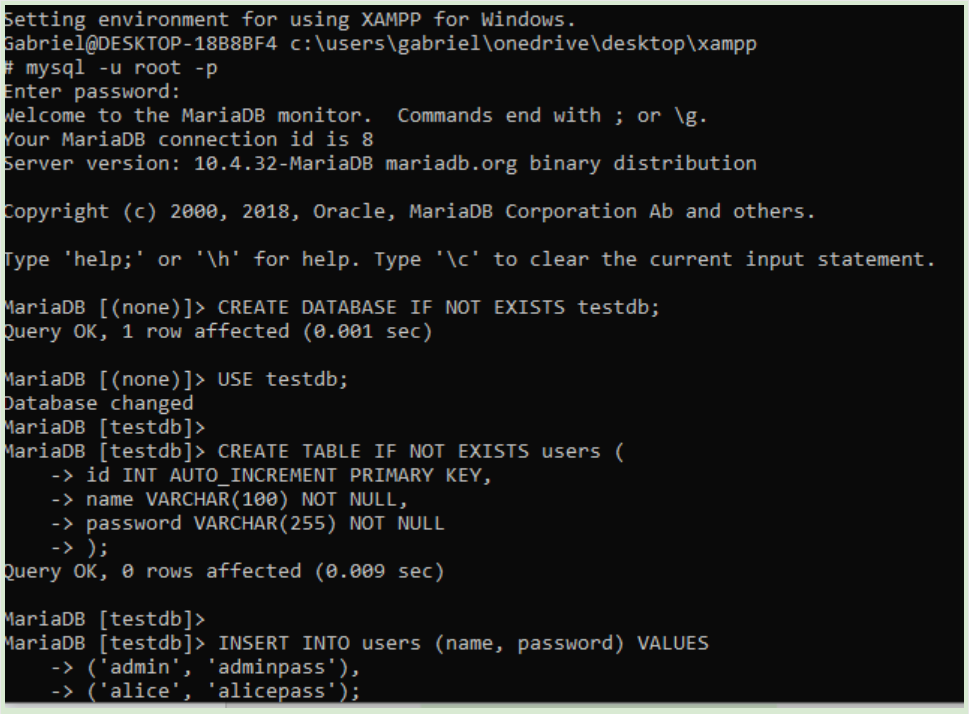
Exemplo Prático (Java + MySQL)

A seguir, uma demonstração completa em Java: criamos um banco de exemplo com usuários e implementamos uma classe vulnerável e outra segura.

SQL (Criação do BD): Primeiro, crie o banco de dados e a tabela em MySQL/PHPMyAdmin, recomendo fazer pelo XAMPP:

No XAMPP: `mysql -u root -p`
depois dê enter quando pedir senha e cole (apenas clicando com botão direito):

```
CREATE DATABASE IF NOT EXISTS testdb;
USE testdb;

CREATE TABLE IF NOT EXISTS users (
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(100) NOT NULL,
password VARCHAR(255) NOT NULL
);

INSERT INTO users (name, password) VALUES
('admin', 'adminpass'),
('alice', 'alicepass');
```

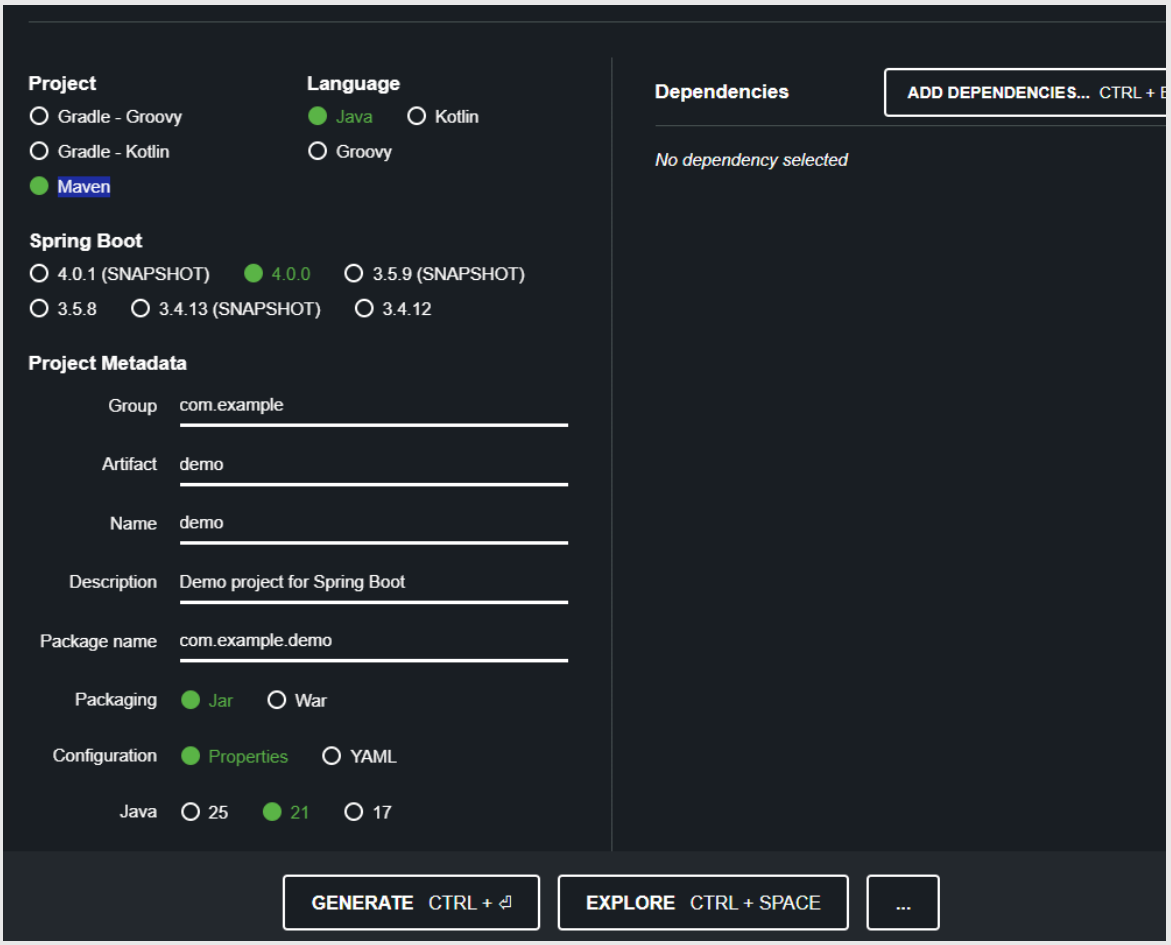


Configuração do Driver JDBC: Adicione o driver MySQL ao projeto e outros que usar. Se usar Maven, inclua no *pom.xml as dependências*:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>4.0.0</version>
      <relativePath/>
```

```
    </parent>

    <groupId>com.example</groupId>
    <artifactId>exemploIot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>exemploIot</name>
    <description>Demo project for Spring Boot - Injeção SQL</description>

    <properties>
        <java.version>21</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- Para Thymeleaf (interface web opcional) -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Se não usar Maven, baixe cada dependência usada pesquisando-as e coloque no jar do projeto, é bem mais chato, use Maven, pesquise Spring Initializr (https://start.spring.io) e vai na fé.



Código Java Vulnerável (VulnerableExample.java): Este programa recebe o nome de usuário pela linha de comando e consulta o banco vulneravelmente, teste `' OR '1'='1` ao logar:

```
package com.example.exemploIot;

import java.sql.*;
import java.util.Scanner;

public class VulnerableExample {
   // SENHA VAZIA - como descobrimos no teste
   private static final String DB_URL =

"jdbc:mysql://localhost:3306/testdb?useSSL=false&serverTimezone=UTC";
```

```
    private static final String DB_USER = "root";
    private static final String DB_PASS = "";  // SENHA VAZIA!

    public static void main(String[] args) {
        System.out.println("========================================");
        System.out.println("   DEMONSTRAÇÃO DE INJEÇÃO SQL");
        System.out.println("   (Código VULNERÁVEL - Não use em produção!)");
        System.out.println("========================================");
        System.out.println("Credenciais detectadas:");
        System.out.println("  Usuário: " + DB_USER);
        System.out.println("  Senha: [VAZIA]");
        System.out.println("========================================");

        // Primeiro, verificar se o banco existe
        if (!verificarBanco()) {
            System.out.println("\n⚠  Banco de dados não encontrado!");
            System.out.println("Executando script de criação...");
            criarBancoETabela();
        } else {
            System.out.println("\n✅ Banco de dados encontrado!");
            mostrarUsuariosExistentes();
        }

        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n" + "=".repeat(50));
            System.out.println("          MENU PRINCIPAL");
            System.out.println("=".repeat(50));
            System.out.println("1. Testar login normal");
            System.out.println("2. Testar ataque SQL Injection");
            System.out.println("3. Digitar query manualmente");
            System.out.println("4. Ver todos os usuários");
            System.out.println("5. Adicionar novo usuário");
            System.out.println("6. Executar comando SQL personalizado");
            System.out.println("0. Sair");
            System.out.print("Escolha uma opção: ");

            String opcao = scanner.nextLine();

            switch (opcao) {
                case "1":
                    testarLoginNormal(scanner);
                    break;
                case "2":
                    testarAtaqueInjecao(scanner);
                    break;
                case "3":
                    testarManual(scanner);
                    break;
                case "4":
                    mostrarTodosUsuarios();
                    break;
```

```
            case "5":
                adicionarUsuario(scanner);
                break;
            case "6":
                executarSQLPersonalizado(scanner);
                break;
            case "0":
                System.out.println("Saindo...");
                scanner.close();
                return;
            default:
                System.out.println("Opção inválida!");
        }
    }
}

private static boolean verificarBanco() {
    try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS)) {
        // Verificar se a tabela existe
        String sql = "SHOW TABLES LIKE 'users'";
        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
            return rs.next();
        }
    } catch (SQLException e) {
        System.out.println("Erro ao verificar banco: " + e.getMessage());
        return false;
    }
}

private static void criarBancoETabela() {
    try (Connection conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/?useSSL=false&serverTimezone=UTC",
            DB_USER, DB_PASS);
         Statement stmt = conn.createStatement()) {

        // Criar banco de dados se não existir
        stmt.execute("CREATE DATABASE IF NOT EXISTS testdb");
        stmt.execute("USE testdb");

        // Criar tabela
        String createTable = "CREATE TABLE IF NOT EXISTS users (" +
                "id INT AUTO_INCREMENT PRIMARY KEY, " +
                "name VARCHAR(100) NOT NULL, " +
                "password VARCHAR(255) NOT NULL)";
        stmt.execute(createTable);

        // Inserir dados de exemplo
        String insertData = "INSERT INTO users (name, password) VALUES " +
                "('admin', 'admin123'), " +
                "('alice', 'alice123'), " +
                "('bob', 'bob123'), " +
```

```
                "('charlie', 'charlie123')";
            stmt.executeUpdate(insertData);

            System.out.println("✅ Banco 'testdb' criado com sucesso!");
            System.out.println("✅ Tabela 'users' criada com 4 usuários de
exemplo");

        } catch (SQLException e) {
            System.out.println("Erro ao criar banco: " + e.getMessage());
        }
    }

    private static void mostrarUsuariosExistentes() {
        System.out.println("\nUsuários existentes no banco:");
        System.out.println("-".repeat(30));

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery("SELECT * FROM users")) {

            int count = 0;
            while (rs.next()) {
                count++;
                System.out.printf("%d. %-10s (senha: %s)%n",
                        rs.getInt("id"),
                        rs.getString("name"),
                        rs.getString("password"));
            }

            if (count == 0) {
                System.out.println("Nenhum usuário encontrado!");
            } else {
                System.out.println("-".repeat(30));
                System.out.println("Total: " + count + " usuário(s)");
            }

        } catch (SQLException e) {
            System.out.println("Erro ao buscar usuários: " + e.getMessage());
        }
    }

    private static void testarLoginNormal(Scanner scanner) {
        System.out.println("\n" + "-".repeat(50));
        System.out.println("          LOGIN NORMAL");
        System.out.println("-".repeat(50));
        System.out.println("Digite um nome de usuário válido:");
        System.out.println("Exemplos: admin, alice, bob, charlie");
        System.out.print("Usuário: ");
        String username = scanner.nextLine();

        executarQueryVulneravel(username, "Login normal");
    }
```

```java
private static void testarAtaqueInjecao(Scanner scanner) {
    System.out.println("\n" + "=".repeat(50));
    System.out.println("       ATAQUE SQL INJECTION");
    System.out.println("=".repeat(50));
    System.out.println("Escolha o tipo de ataque:");
    System.out.println("1. ' OR '1'='1 (retorna TODOS os usuários)");
    System.out.println("2. admin' -- (ignora verificação de senha)");
    System.out.println("3. ' UNION SELECT... (injetar dados falsos)");
    System.out.println("4. '; DROP TABLE users; -- (ATAQUE DESTRUTIVO!)");
    System.out.println("5. Outro (digitar manualmente)");
    System.out.print("Opção: ");

    String tipo = scanner.nextLine();
    String username = "";
    String descricao = "";

    switch (tipo) {
        case "1":
            username = "' OR '1'='1";
            descricao = "Ataque básico: sempre verdadeiro";
            break;
        case "2":
            username = "admin' -- ";
            descricao = "Bypass de autenticação";
            break;
        case "3":
            username = "' UNION SELECT 99, 'hacker', 'pwned' -- ";
            descricao = "Injeção com UNION";
            break;
        case "4":
            username = "'; DROP TABLE users; -- ";
            descricao = "ATAQUE DESTRUTIVO - APAGA TABELA!";
            System.out.println("\n⚠  ⚠  ⚠  ALERTA CRÍTICO!");
            System.out.println("Este comando vai APAGAR a tabela users!");
            System.out.print("Tem certeza? (s/N): ");
            if (!scanner.nextLine().equalsIgnoreCase("s")) {
                System.out.println("Operação cancelada.");
                return;
            }
            break;
        case "5":
            System.out.print("Digite o payload: ");
            username = scanner.nextLine();
            descricao = "Ataque personalizado";
            break;
        default:
            System.out.println("Opção inválida!");
            return;
    }

    executarQueryVulneravel(username, "Ataque: " + descricao);
}
```

```
   private static void testarManual(Scanner scanner) {
      System.out.println("\n" + "=".repeat(50));
      System.out.println("       TESTE MANUAL");
      System.out.println("=".repeat(50));
      System.out.println("Digite qualquer input para testar:");
      System.out.println("Exemplo: admin' OR 'a'='a");
      System.out.print("Input: ");
      String input = scanner.nextLine();

      executarQueryVulneravel(input, "Teste manual");
   }

   private static void executarQueryVulneravel(String userInput, String
tipoTeste) {
      // CÓDIGO VULNERÁVEL - CONCATENAÇÃO DIRETA
      String sql = "SELECT * FROM users WHERE name = '" + userInput + "'";

      System.out.println("\n" + "=".repeat(60));
      System.out.println("TIPO: " + tipoTeste);
      System.out.println("=".repeat(60));
      System.out.println("SQL GERADO:");
      System.out.println("  " + sql);
      System.out.println("=".repeat(60));

      try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
           Statement stmt = conn.createStatement();
           ResultSet rs = stmt.executeQuery(sql)) {

         System.out.println("\n📊 RESULTADOS:");
         System.out.println("-".repeat(40));

         int count = 0;
         while (rs.next()) {
            count++;
            System.out.println("Usuário #" + count + ":");
            System.out.println("  ID:       " + rs.getInt("id"));
            System.out.println("  Nome:     " + rs.getString("name"));
            System.out.println("  Senha:    " + rs.getString("password"));
            System.out.println("-".repeat(40));
         }

         System.out.println("\n📈 RESUMO DA EXECUÇÃO:");
         System.out.println("-".repeat(40));

         if (count == 0) {
            System.out.println("❌ Nenhum usuário encontrado");
            System.out.println("   Input não corresponde a nenhum usuário");
         } else if (count == 1) {
            System.out.println("✅ Login bem-sucedido!");
            System.out.println("   Apenas 1 usuário retornado (comportamento
normal)");
```

```
            } else {
                System.out.println("⚠  ⚠  ⚠  ALERTA DE SEGURANÇA!");
                System.out.println("   " + count + " usuários retornados!");
                System.out.println("   Possível SQL Injection detectado!");
                System.out.println("   O atacante conseguiu acesso a múltiplos
usuários!");
            }

            // Detectar se é um ataque
            if (userInput.contains("' OR") || userInput.contains("--") ||
                  userInput.contains("UNION") || userInput.contains(";")) {
                System.out.println("\n🔴 INDÍCIOS DE ATAQUE DETECTADOS:");
                if (userInput.contains("' OR")) System.out.println("  • Possível
condição OR maliciosa");
                if (userInput.contains("--")) System.out.println("  • Comentário
SQL detectado");
                if (userInput.contains("UNION")) System.out.println("  • Injeção
UNION detectada");
                if (userInput.contains(";")) System.out.println("  • Múltiplos
comandos detectados");
            }

        } catch (SQLException e) {
            System.out.println("\n💥 ERRO NA EXECUÇÃO:");
            System.out.println("  " + e.getMessage());

            if (e.getMessage().contains("syntax")) {
                System.out.println("\n⚠  Erro de sintaxe no SQL!");
                System.out.println("  O input do usuário corrompeu a query
SQL.");
            }

            if (e.getMessage().contains("Table") &&
e.getMessage().contains("doesn't exist")) {
                System.out.println("\n💀 ATAQUE DESTRUTIVO BEM-SUCEDIDO!");
                System.out.println("  A tabela foi apagada pelo ataque!");
                System.out.println("  Execute a opção 'Recriar banco' no menu.");
            }
        }
    }

    private static void mostrarTodosUsuarios() {
        System.out.println("\n" + "—".repeat(50));
        System.out.println("       TODOS OS USUÁRIOS");
        System.out.println("—".repeat(50));

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery("SELECT * FROM users ORDER BY
id")) {

            System.out.printf("%-5s %-15s %-15s%n", "ID", "NOME", "SENHA");
```

```
        System.out.println("-".repeat(40));

        int total = 0;
        while (rs.next()) {
           total++;
           System.out.printf("%-5d %-15s %-15s%n",
                 rs.getInt("id"),
                 rs.getString("name"),
                 rs.getString("password"));
        }

        System.out.println("-".repeat(40));
        System.out.println("Total: " + total + " usuário(s)");

     } catch (SQLException e) {
        System.out.println("Erro: " + e.getMessage());
     }
  }

  private static void adicionarUsuario(Scanner scanner) {
     System.out.println("\n" + "-".repeat(50));
     System.out.println("       ADICIONAR USUÁRIO");
     System.out.println("-".repeat(50));

     System.out.print("Nome do usuário: ");
     String nome = scanner.nextLine();

     System.out.print("Senha: ");
     String senha = scanner.nextLine();

     // Método SEGURO usando PreparedStatement
     String sql = "INSERT INTO users (name, password) VALUES (?, ?)";

     try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
          PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString(1, nome);
        pstmt.setString(2, senha);

        int rows = pstmt.executeUpdate();
        System.out.println("\n✅ Usuário '" + nome + "' adicionado com
sucesso!");
        System.out.println("   (Método seguro - PreparedStatement)");

     } catch (SQLException e) {
        System.out.println("Erro: " + e.getMessage());
     }
  }

  private static void executarSQLPersonalizado(Scanner scanner) {
     System.out.println("\n" + "-".repeat(50));
     System.out.println("       SQL PERSONALIZADO");
```

```
      System.out.println("=".repeat(50));
      System.out.println("CUIDADO: Esta opção permite executar qualquer
SQL!");
      System.out.println("Pode ser usado para demonstrar comandos
perigosos.");
      System.out.print("Digite o comando SQL: ");

      String sql = scanner.nextLine();

      System.out.println("\nExecutando: " + sql);
      System.out.println("=".repeat(50));

      try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
           Statement stmt = conn.createStatement()) {

         if (sql.trim().toUpperCase().startsWith("SELECT")) {
            // É uma consulta
            ResultSet rs = stmt.executeQuery(sql);

            System.out.println("\nResultados:");
            System.out.println("-".repeat(40));

            int count = 0;
            while (rs.next()) {
               count++;
               // Mostrar colunas disponíveis
               System.out.println("Linha #" + count + ":");
               for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
                  System.out.printf("  %s: %s%n",
                        rs.getMetaData().getColumnName(i),
                        rs.getString(i));
               }
               System.out.println("-".repeat(30));
            }

            System.out.println("Total de linhas: " + count);

         } else {
            // É um comando de atualização
            int rows = stmt.executeUpdate(sql);
            System.out.println("✅ Comando executado!");
            System.out.println("Linhas afetadas: " + rows);
         }

      } catch (SQLException e) {
         System.out.println("💥 ERRO: " + e.getMessage());
      }
   }
}
```

Teste com entrada maliciosa no login:

`' OR '1'='1`

A query montada será `SELECT * FROM users WHERE name = '' OR '1'='1'`. Como explicamos, `'1'='1'` é sempre verdadeiro, então a consulta retorna usuários (por exemplo, `admin`) e o programa logará sempre, sem senha.

No codigo a seguir, rode e teste `' OR '1'='1` ao logar

Código Java Seguro (SecureExample.java) – usando PreparedStatement:

```
package com.example.exemploIot;

import java.sql.*;
import java.util.Scanner;

public class SecureExample {
   // Credenciais (senha vazia como descoberto)
   private static final String DB_URL =

"jdbc:mysql://localhost:3306/testdb?useSSL=false&serverTimezone=UTC";
   private static final String DB_USER = "root";
   private static final String DB_PASS = "";

   public static void main(String[] args) {

System.out.println("╔═══════════════════════════════════════════════════════╗");
       System.out.println("║     DEMONSTRAÇÃO SEGURA (PROTEGIDA)
║");
       System.out.println("║     Usando PreparedStatement
║");

System.out.println("╚═══════════════════════════════════════════════════════╝");

       // Verificar/Criar banco
       inicializarBanco();

       Scanner scanner = new Scanner(System.in);

       while (true) {
           System.out.println("\n" + "=".repeat(60));
           System.out.println("           MENU - VERSÃO SEGURA");
           System.out.println("=".repeat(60));
           System.out.println("1. Login seguro (PreparedStatement)");
           System.out.println("2. Buscar usuários com filtro seguro");
           System.out.println("3. Adicionar usuário (protegido)");
           System.out.println("4. Atualizar senha (protegido)");
           System.out.println("5. Comparar com versão vulnerável");
           System.out.println("6. Testar ataques (serão bloqueados)");
```

```
            System.out.println("7. Ver todos os usuários");
            System.out.println("0. Sair");
            System.out.print("Escolha: ");

            String opcao = scanner.nextLine();

            switch (opcao) {
                case "1": loginSeguro(scanner); break;
                case "2": buscarUsuariosSeguro(scanner); break;
                case "3": adicionarUsuarioSeguro(scanner); break;
                case "4": atualizarSenhaSeguro(scanner); break;
                case "5": compararVersoes(scanner); break;
                case "6": testarAtaquesBloqueados(scanner); break;
                case "7": mostrarTodosUsuarios(); break;
                case "0":
                    System.out.println("Encerrando...");
                    scanner.close();
                    return;
                default: System.out.println("Opção inválida!");
            }
        }
    }

    private static void inicializarBanco() {
        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS)) {
            System.out.println("\n✅ Conectado ao MySQL!");

            // Criar tabela se não existir
            String createTable = "CREATE TABLE IF NOT EXISTS users (" +
                    "id INT AUTO_INCREMENT PRIMARY KEY, " +
                    "name VARCHAR(100) NOT NULL, " +
                    "password VARCHAR(255) NOT NULL, " +
                    "email VARCHAR(255), " +
                    "created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP)";

            try (Statement stmt = conn.createStatement()) {
                stmt.execute(createTable);

                // Verificar se há dados
                ResultSet rs = stmt.executeQuery("SELECT COUNT(*) FROM
users");
                rs.next();
                int count = rs.getInt(1);

                if (count == 0) {
                    // Inserir dados iniciais
                    String insert = "INSERT INTO users (name, password, email)
VALUES " +
                            "('admin', 'admin123', 'admin@exemplo.com'), " +
                            "('alice', 'alice123', 'alice@exemplo.com'), " +
                            "('bob', 'bob123', 'bob@exemplo.com')";
                    stmt.executeUpdate(insert);
```

```
                    System.out.println("✅ 3 usuários de exemplo criados!");
                } else {
                    System.out.println("✅ Tabela 'users' já existe com " +
count + " registro(s)");
                }
            }

        } catch (SQLException e) {
            System.out.println("❌ Erro ao inicializar banco: " +
e.getMessage());
        }
    }

    // ========== MÉTODO 1: LOGIN SEGURO ==========
    private static void loginSeguro(Scanner scanner) {
        System.out.println("\n" + "=".repeat(50));
        System.out.println("          LOGIN SEGURO");
        System.out.println("=".repeat(50));

        System.out.print("Nome de usuário: ");
        String username = scanner.nextLine();

        System.out.print("Senha: ");
        String password = scanner.nextLine();

        // QUERY SEGURA - PreparedStatement
        String sql = "SELECT * FROM users WHERE name = ? AND password = ?";

        System.out.println("\n🔒 SQL com PreparedStatement:");
        System.out.println("   " + sql);
        System.out.println("   Parâmetros: [1] = \"" + username + "\", [2] =
\"" + password + "\"");

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            // Definir parâmetros (isso é SEGURO!)
            pstmt.setString(1, username);
            pstmt.setString(2, password);

            System.out.println("\n📦 Dados enviados ao banco:");
            System.out.println("   • Query fixa: SELECT * FROM users WHERE name
= ? AND password = ?");
            System.out.println("   • Parâmetro 1 (nome): \"" + username +
"\"");
            System.out.println("   • Parâmetro 2 (senha): \"" + password +
"\"");
            System.out.println("   ⚡ O banco NUNCA mistura parâmetros com
código SQL!");

            try (ResultSet rs = pstmt.executeQuery()) {
                System.out.println("\n📊 RESULTADO:");
```

```
                System.out.println("-".repeat(40));

                if (rs.next()) {
                    System.out.println("✅ LOGIN BEM-SUCEDIDO!");
                    System.out.println("   ID: " + rs.getInt("id"));
                    System.out.println("   Nome: " + rs.getString("name"));
                    System.out.println("   Email: " + rs.getString("email"));
                    System.out.println("   Criado em: " +
rs.getTimestamp("created_at"));
                } else {
                    System.out.println("❌ Credenciais inválidas!");
                    System.out.println("   Nenhum usuário encontrado com essas
credenciais.");
                }
            }

        } catch (SQLException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }

    // ========== MÉTODO 2: BUSCA SEGURA ==========
    private static void buscarUsuariosSeguro(Scanner scanner) {
        System.out.println("\n" + "-".repeat(50));
        System.out.println("          BUSCA SEGURA (LIKE)");
        System.out.println("-".repeat(50));

        System.out.print("Digite parte do nome para buscar: ");
        String filtro = scanner.nextLine();

        // SEGURO: PreparedStatement com LIKE
        String sql = "SELECT * FROM users WHERE name LIKE ? ORDER BY name";

        System.out.println("\n🔒 SQL seguro com LIKE:");
        System.out.println("   " + sql);
        System.out.println("   Parâmetro: [1] = \"%" + filtro + "%\"");

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, "%" + filtro + "%");

            try (ResultSet rs = pstmt.executeQuery()) {
                System.out.println("\n🔍 Resultados da busca:");

System.out.println("┌──────┬──────────────┬──────────────────────────┬──────────────
──────┐");
                System.out.println("| ID   | Nome         |       Email              |
Criado em        |");

System.out.println("├──────┼──────────────┼──────────────────────────┼──────────────
──────┤");
```

```
                int count = 0;
                while (rs.next()) {
                    count++;
                    System.out.printf("| %-3d | %-8s | %-20s | %-19s |%n",
                            rs.getInt("id"),
                            rs.getString("name"),
                            rs.getString("email"),

rs.getTimestamp("created_at").toString().substring(0, 19));
                }


System.out.println("└─────┴──────────┴──────────────────────┴─────────────────
──────┘");
                System.out.println("Total encontrado: " + count + "
usuário(s)");
            }

        } catch (SQLException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }

    // ========== MÉTODO 3: ADIÇÃO SEGURA ==========
    private static void adicionarUsuarioSeguro(Scanner scanner) {
        System.out.println("\n" + "─".repeat(50));
        System.out.println("          ADICIONAR USUÁRIO (SEGURO)");
        System.out.println("─".repeat(50));

        System.out.print("Nome: ");
        String nome = scanner.nextLine();

        System.out.print("Senha: ");
        String senha = scanner.nextLine();

        System.out.print("Email: ");
        String email = scanner.nextLine();

        // SEGURO: INSERT com PreparedStatement
        String sql = "INSERT INTO users (name, password, email) VALUES (?, ?,
?)";

        System.out.println("\n🔒 SQL seguro para INSERT:");
        System.out.println("   " + sql);
        System.out.println("   Parâmetros: [1]=\"" + nome + "\", [2]=\"" +
senha + "\", [3]=\"" + email + "\"");

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, nome);
```

```
            pstmt.setString(2, senha);
            pstmt.setString(3, email);

            int linhasAfetadas = pstmt.executeUpdate();

            System.out.println("\n✅ USUÁRIO ADICIONADO COM SUCESSO!");
            System.out.println("   Linhas afetadas: " + linhasAfetadas);
            System.out.println("\n🔐 POR QUE É SEGURO?");
            System.out.println("   1. A query é PRÉ-COMPILADA no banco");
            System.out.println("   2. Os valores são passados como
PARÂMETROS");
            System.out.println("   3. O banco NUNCA mistura valores com
código");
            System.out.println("   4. Mesmo valores maliciosos são tratados
como DADOS");

        } catch (SQLException e) {
            System.out.println("❌ Erro: " + e.getMessage());

            if (e.getMessage().contains("Duplicate entry")) {
                System.out.println("   Já existe um usuário com esse nome!");
            }
        }
    }

    // ========== MÉTODO 4: ATUALIZAÇÃO SEGURA ==========
    private static void atualizarSenhaSeguro(Scanner scanner) {
        System.out.println("\n" + "=".repeat(50));
        System.out.println("          ATUALIZAR SENHA (SEGURO)");
        System.out.println("=".repeat(50));

        System.out.print("Nome do usuário: ");
        String nome = scanner.nextLine();

        System.out.print("Nova senha: ");
        String novaSenha = scanner.nextLine();

        // SEGURO: UPDATE com PreparedStatement
        String sql = "UPDATE users SET password = ? WHERE name = ?";

        System.out.println("\n🔒 SQL seguro para UPDATE:");
        System.out.println("   " + sql);
        System.out.println("   Parâmetros: [1]=\"" + novaSenha + "\", [2]=\"" +
nome + "\"");

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, novaSenha);
            pstmt.setString(2, nome);

            int linhasAfetadas = pstmt.executeUpdate();
```

```
            if (linhasAfetadas > 0) {
                System.out.println("\n✅ SENHA ATUALIZADA COM SUCESSO!");
                System.out.println("   Usuário: " + nome);
                System.out.println("   Linhas afetadas: " + linhasAfetadas);
            } else {
                System.out.println("\n⚠  Nenhum usuário encontrado com nome: 
" + nome);
            }

        } catch (SQLException e) {
            System.out.println("❌ Erro: " + e.getMessage());
        }
    }

    // ========== MÉTODO 5: COMPARAÇÃO ==========
    private static void compararVersoes(Scanner scanner) {
        System.out.println("\n" + "=".repeat(60));
        System.out.println("          COMPARAÇÃO: VULNERÁVEL vs SEGURO");
        System.out.println("=".repeat(60));

        System.out.println("\n📝 MESMO INPUT: \"' OR '1'='1\"");
        System.out.println("-".repeat(40));

        System.out.println("\n🔴 VERSÃO VULNERÁVEL (Statement):");
        System.out.println("   String sql = \"SELECT * FROM users WHERE name = 
'\" + input + \"'\";");
        System.out.println("   SQL gerado: SELECT * FROM users WHERE name = '' 
OR '1'='1'");
        System.out.println("   ⚡ RESULTADO: Retorna TODOS os usuários!");
        System.out.println("   💥 ATAQUE BEM-SUCEDIDO!");

        System.out.println("\n🟢 VERSÃO SEGURA (PreparedStatement):");
        System.out.println("   String sql = \"SELECT * FROM users WHERE name = 
?\";");
        System.out.println("   pstmt.setString(1, \"' OR '1'='1\");");
        System.out.println("   ⚡ O banco executa: SELECT * FROM users WHERE 
name = \"' OR '1'='1\"");
        System.out.println("   ✅ RESULTADO: Busca usuário com nome \"' OR 
'1'='1\"");
        System.out.println("   ✅ Nenhum ataque - Apenas busca por texto!");

        System.out.println("\n🎯 DIFERENÇA CHAVE:");
        System.out.println("   Vulnerável: Dados + Código misturados");
        System.out.println("   Seguro:    Dados SEPARADOS do Código");

        System.out.print("\nPressione Enter para ver demonstração 
prática...");
        scanner.nextLine();

        // Demonstração prática
        demonstrarComparacao();
    }
```

```
    private static void demonstrarComparacao() {
        String inputMalicioso = "' OR '1'='1";

        System.out.println("\n🧪 TESTE PRÁTICO COM INPUT: \"" +
inputMalicioso + "\"");

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS)) {

            // Teste 1: Vulnerável
            System.out.println("\n1. 🔴 MÉTODO VULNERÁVEL:");
            String sqlVulneravel = "SELECT * FROM users WHERE name = '" +
inputMalicioso + "'";
            System.out.println("   SQL: " + sqlVulneravel);

            try (Statement stmt = conn.createStatement();
                 ResultSet rs = stmt.executeQuery(sqlVulneravel)) {

                int count = 0;
                while (rs.next()) { count++; }
                System.out.println("   Resultado: " + count + " usuário(s)
retornados");
                if (count > 1) {
                    System.out.println("   💥 VULNERABILIDADE EXPLORADA!");
                }
            }

            // Teste 2: Seguro
            System.out.println("\n2. 🟢 MÉTODO SEGURO:");
            String sqlSeguro = "SELECT * FROM users WHERE name = ?";
            System.out.println("   SQL: " + sqlSeguro);
            System.out.println("   Parâmetro: \"" + inputMalicioso + "\"");

            try (PreparedStatement pstmt = conn.prepareStatement(sqlSeguro)) {
                pstmt.setString(1, inputMalicioso);

                try (ResultSet rs = pstmt.executeQuery()) {
                    int count = 0;
                    while (rs.next()) { count++; }
                    System.out.println("   Resultado: " + count + " usuário(s)
retornados");
                    System.out.println("   ✅ SEGURO - Apenas busca texto
literal!");
                }
            }

        } catch (SQLException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }

    // ========== MÉTODO 6: TESTAR ATAQUES BLOQUEADOS ==========
```

```
   private static void testarAtaquesBloqueados(Scanner scanner) {
       System.out.println("\n" + "=".repeat(60));
       System.out.println("          TESTANDO ATAQUES (SERÃO BLOQUEADOS)");
       System.out.println("=".repeat(60));

       String[] ataques = {
               "' OR '1'='1",
               "admin' -- ",
               "'; DROP TABLE users; -- ",
               "' UNION SELECT 1, 'hacker', 'pwned', 'hack@evil.com', NOW()
-- ",
               "' OR name LIKE '%a%' -- ",
               "x' OR 'x'='x"
       };

       String[] descricoes = {
               "Retorna todos os usuários",
               "Bypass de autenticação",
               "Apagar tabela (destrutivo)",
               "Injetar usuário falso",
               "Busca com LIKE malicioso",
               "Condição sempre verdadeira"
       };

       for (int i = 0; i < ataques.length; i++) {
           System.out.println("\n🔴 TESTE " + (i+1) + ": " + descricoes[i]);
           System.out.println("   Input: \"" + ataques[i] + "\"");

           testarAtaqueIndividual(ataques[i]);

           System.out.print("\nPressione Enter para próximo teste...");
           scanner.nextLine();
       }

       System.out.println("\n✅ CONCLUSÃO: Todos os ataques foram
BLOQUEADOS!");
       System.out.println("   PreparedStatement trata entrada como DADO, não
como CÓDIGO");
   }

   private static void testarAtaqueIndividual(String ataque) {
       String sql = "SELECT * FROM users WHERE name = ?";

       try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
            PreparedStatement pstmt = conn.prepareStatement(sql)) {

           pstmt.setString(1, ataque);

           try (ResultSet rs = pstmt.executeQuery()) {
               int count = 0;
               while (rs.next()) { count++; }
```

```
                if (count == 0) {
                    System.out.println("   ✅ RESULTADO: Nenhum usuário
encontrado");
                    System.out.println("      (O ataque foi tratado como busca
por texto)");
                } else {
                    System.out.println("   ⚠  RESULTADO: " + count + "
usuário(s)");
                    System.out.println("      (Coincidência normal, não é
injeção)");
                }
            }

        } catch (SQLException e) {
            System.out.println("   ❌ Erro: " + e.getMessage());
        }
    }

    // ========== MÉTODO 7: VER TODOS OS USUÁRIOS ==========
    private static void mostrarTodosUsuarios() {
        System.out.println("\n" + "=".repeat(50));
        System.out.println("          TODOS OS USUÁRIOS");
        System.out.println("=".repeat(50));

        // Mesmo para SELECT simples, use PreparedStatement se houver filtros
        String sql = "SELECT * FROM users ORDER BY id";

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASS);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {


System.out.println("┌──────┬────────────┬──────────────┬──────────────────────────┬
──────────────────────┐");
            System.out.println("| ID   | Nome     | Senha       |        Email
|     Criado em       |");

System.out.println("├──────┼────────────┼──────────────┼──────────────────────────┼
──────────────────────┤");

            int total = 0;
            while (rs.next()) {
                total++;
                System.out.printf("| %-3d | %-8s | %-12s | %-20s | %-19s |%n",
                        rs.getInt("id"),
                        rs.getString("name"),
                        "••••••••",
                        rs.getString("email"),
                        rs.getTimestamp("created_at").toString().substring(0,
19));
            }
```

```
System.out.println("└─────┴──────────┴──────────────┴──────────────────────────┘");
            System.out.println("Total: " + total + " usuário(s)");

        } catch (SQLException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

Teste com a mesma entrada `' OR '1'='1`:

Desta vez o driver JDBC tratará `userInput` como dado. A query efetivamente procurará um nome que seja literalmente `" ' OR '1'='1 "`. Como não há usuário com esse nome, nenhum resultado é retornado e o login falha normalmente. A injeção não surte efeito, comprovando que o ataque foi barrado.

Sistema normal de login:

```
SELECT * FROM users WHERE username = 'admin' AND password = '123'
```

Atacante insere:

```
Usuário: ' OR '1'='1
```

```
Senha: qualquer coisa
```

SQL gerado:

```
SELECT * FROM users WHERE username = '' OR '1'='1'
```

O que acontece:

1. `--` comenta o resto do SQL
2. A verificação de senha `AND password = 'qualquer'` é ignorada
3. A query retorna todos os usuários
4. O sistema pensa: "Ah, encontrou um usuário! Login válido!"
5. O atacante loga SEM saber senha.

Resumo

A prevenção de injeção de SQL (e NoSQL) baseia-se em quatro práticas-chave:
Evitar consultas dinâmicas com concatenação de entradas não confiáveis;
Usar consultas parametrizadas (PreparedStatement/ORM) para separar código de dados;
Validar rigorosamente as entradas (whitelist) antes de usá-las;

Conceder apenas os privilégios mínimos necessários ao usuário de banco. Essas medidas garantem que todas as entradas dos usuários sejam tratadas como dados literais, bloqueando tentativas de injeção. Seguindo essas boas práticas (e adotando *stored procedures* seguras quando aplicável), o sistema fica protegido contra a alteração maliciosa das consultas SQL.

Fontes e Ferramentas: Informações adaptadas de documentação e guias reconhecidos (MDNdeveloper.mozilla.org;
OWASPcheatsheetseries.owasp.orgcheatsheetseries.owasp.orgcheatsheetseries.owasp.org;
Impervaimperva.comimperva.com
ChatGPT (Usado de metodologia para pesquisa em sites, usado para estilizar e melhorar o documento e criar códigos para mim)).

Documento criador por:
Gabriel Karkotli Henriques.