CS 405 Module Two: SQL Injection Activity Summary  
Name: Christian Busca  
Course: CS 405 – Secure Coding  
Instructor: Inshan Singh  
Date: July 13, 2025

### **Overview**

As part of my role in this assignment, I acted as a senior developer addressing a SQL injection vulnerability in a simulated banking web application. The development team had discovered unauthorized data access occurring through manipulated SQL queries that exposed sensitive information, including user passwords. The issue was reproduced using a standalone in-memory SQLite database.

The central task was to identify and mitigate this vulnerability—specifically targeting SQL injection attempts using the "OR value=value" pattern. The original implementation was vulnerable because it directly accepted and executed raw SQL strings, making it possible for attackers to modify query logic.

### **Approach and Fix**

To address the vulnerability, I focused on enhancing the run\_query() function. The core fix was to implement a reliable detection mechanism that flags suspected SQL injection patterns and prevents execution when those patterns are identified.

My solution included:

* Scanning the incoming SQL query string for known injection signatures such as " OR " followed by a repeated literal (e.g., 1=1, 'a'='a').
* Rejecting queries containing these patterns and outputting a warning message to the console.
* Preserving functionality for legitimate queries while ensuring malicious ones do not execute.

This approach was intentionally conservative to demonstrate defensive coding practices without altering the structure of the rest of the application.

### **Justification**

The "OR value=value" pattern is a hallmark of SQL injection attacks, especially when used in the WHERE clause to bypass intended access controls. By programmatically identifying these conditions in lowercase-normalized input, the solution effectively blocks attempts to exploit the logic of the SQL query.

Although parameterized queries represent the industry standard for long-term prevention, this assignment focused on analyzing and blocking a specific type of logic-based injection using defensive pattern detection. This targeted detection aligns with the assignment requirements while demonstrating awareness of more robust alternatives.

### **Results**

After applying the fix, the program was executed multiple times, including standard queries and randomized injection attempts generated by the run\_query\_injection() function. The application correctly returned results for legitimate queries and blocked all detected injection attempts, outputting clear warnings to the console. No unauthorized data was retrieved.

### Conclusion

This exercise highlights the importance of identifying and mitigating injection vulnerabilities early in the development process. The completed implementation successfully prevents one of the most common SQL injection patterns while demonstrating secure coding practices. This work reinforces how careful inspection of user input, combined with appropriate validation or pattern detection, can safeguard critical systems against exploitation.

