**Victor Udeh  
CS405 M2-2  
Date:   
  
Summary of SQL Injection Prevention Implementation**

**Approach Taken**: To prevent SQL injection attacks in this assignment, I modified the run\_query() function to detect and prevent potential SQL injection attacks. Specifically, I focused on preventing the "OR value=value" type of attack. This was accomplished using a regular expression to detect suspicious SQL patterns that indicate a possible injection attempt.

In the run\_query() function, I added a regular expression (std::regex) to identify if the SQL query contains injection patterns like "OR 1=1" or other variations that exploit logical OR operators. If such a pattern is detected, the program outputs a message indicating a potential attack and prevents the execution of the query.

The updated code ensures that only valid queries are executed, thereby mitigating the risk of exposing sensitive data through SQL injection.

**Code Changes**:

1. **Added Regular Expression Detection**:
   * I added a regular expression (std::regex sql\_injection\_pattern) to match common SQL injection patterns, specifically those involving logical conditions like "OR value=value".
   * Before executing the SQL query, I used std::regex\_search() to check if the query matches any known injection pattern. If it does, the query is aborted, and a warning is printed to the console.
2. **Comments Added**:
   * Comments were added to document the changes, explaining why and how each modification helps to prevent SQL injection.

**Why This Approach Works**: The approach works because it identifies common SQL injection attempts that involve manipulating the SQL query with logical conditions. By checking for these patterns before executing a query, we can prevent unauthorized access to sensitive information. Regular expressions are particularly effective here because they can flexibly detect different variations of injection attempts.

**Challenges Encountered**:

* **Regular Expression Design**: Designing a regular expression that could effectively detect SQL injection without generating false positives was a key challenge. I addressed this by focusing on logical "OR" conditions commonly used in SQL injection attacks.
* **Linking SQLite**: There were initial challenges with linking the SQLite library (-lsqlite3) and ensuring the correct architecture (arm64) was supported on my system. These issues were resolved by adjusting the compilation flags.

**Console Output**: Below is a screenshot of the console output from running the program:

* The output shows the successful creation of the "USERS" table and the insertion of sample data.
* Legitimate SQL queries return the expected records.
* SQL injection attempts are detected and aborted, with appropriate messages indicating that an attack was prevented.

A screenshot of a computer

Description automatically generated