The provided Python code, while functional, has some potential security vulnerabilities:

**1. SQL Injection:**

* The code directly inserts user-provided input into the SQL query without proper sanitization or parameterization. This makes it vulnerable to SQL injection attacks, where malicious users can inject malicious SQL code into the input to manipulate or compromise the database.

**2. Data Validation:**

* The code lacks input validation to ensure that the user provides valid data types and formats. This could lead to unexpected behavior or errors in the database.

**3. Error Handling:**

* The code doesn't handle potential exceptions like database connection errors or invalid SQL queries. This could lead to the program crashing or exposing sensitive information in error messages.

**4. Sensitive Data Exposure:**

* If the database contains sensitive information, storing it in plain text could be a security risk. Consider using encryption or hashing techniques to protect sensitive data.

To mitigate these vulnerabilities, consider the following:

* **Parameterized Queries:** Use parameterized queries to prevent SQL injection attacks. This involves passing user input as parameters to the SQL query, rather than directly concatenating it into the string.
* **Input Validation:** Implement robust input validation to ensure that user input is in the expected format and data type.
* **Error Handling:** Handle potential exceptions gracefully to prevent unexpected behavior and information leakage.
* **Data Encryption:** Encrypt sensitive data before storing it in the database to protect it from unauthorized access.
* **Secure Coding Practices:** Follow secure coding practices to avoid common vulnerabilities like buffer overflows, cross-site scripting (XSS), and cross-site request forgery (CSRF).

By addressing these security concerns, you can make the code more secure and protect the integrity of the database.