Simulation Report (Individual Version)

Course: SQL Server Development
Database: AdventureWorks2019
Student Name / ID: Yung-Lun Lee

1. Introduction

In this simulation, I developed and tested Dynamic SQL in both secure and vulnerable ways. I created:

1. A secure procedure using parameterized SQL
2. A vulnerable procedure using string concatenation

I also added input validation, error handling, and execution logging.
The goal was to compare how these two approaches behave in terms of security and performance.

2. System Design

2.1 ExecutionLog Table

I used the existing `Reporting.ExecutionLog` table to store:

- Procedure name
- Executed SQL
- Execution time
- Error message
- User and host
- Duration in milliseconds
- Parameter values

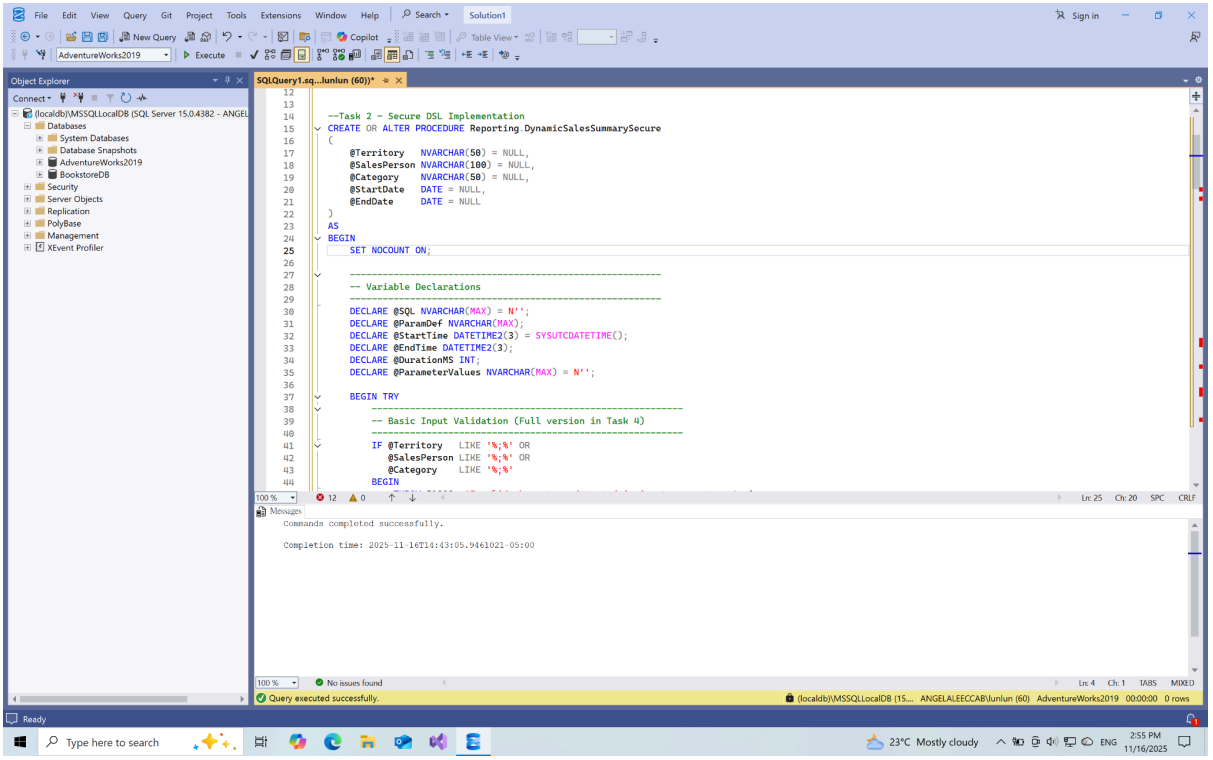
These logs help me review activity, compare results, and identify errors.

2.2 Secure Procedure

The secure procedure:

- Uses `sp_executesql` with parameters
- Prevents SQL injection
- Has TRY/CATCH error handling

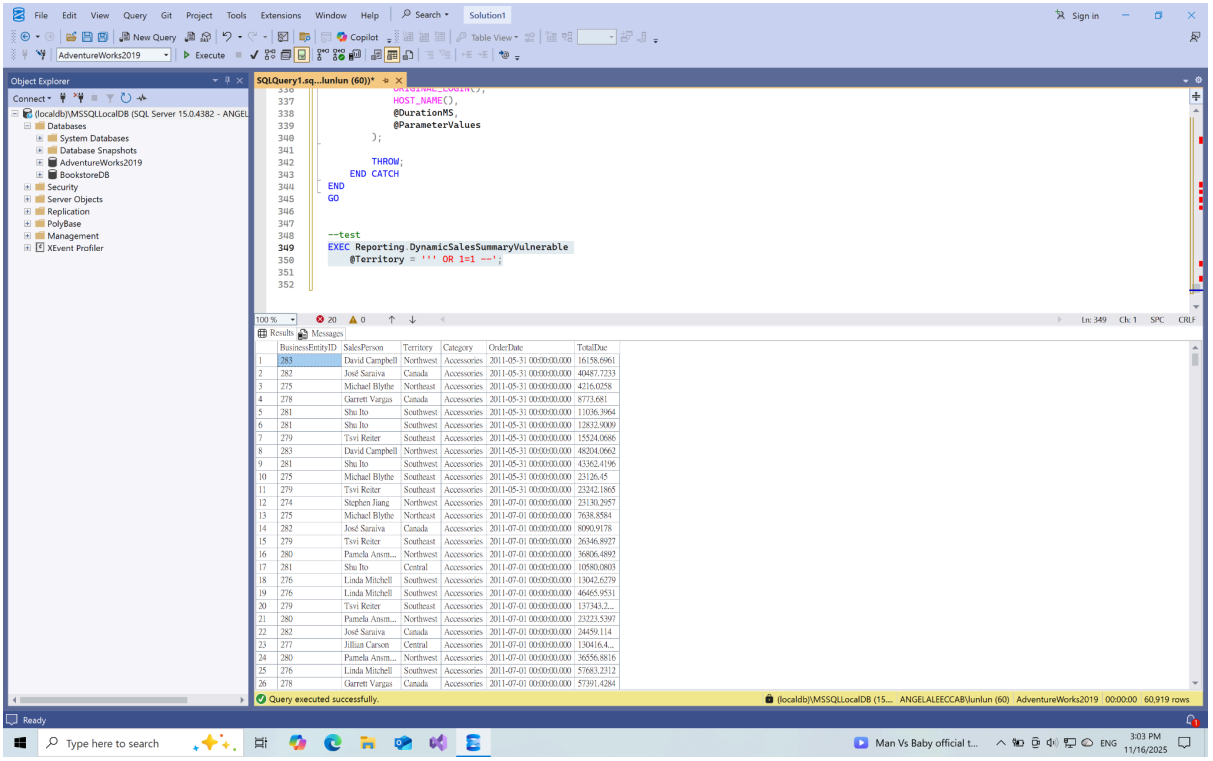
- Logs every execution
- Rejects suspicious input (Task 4)

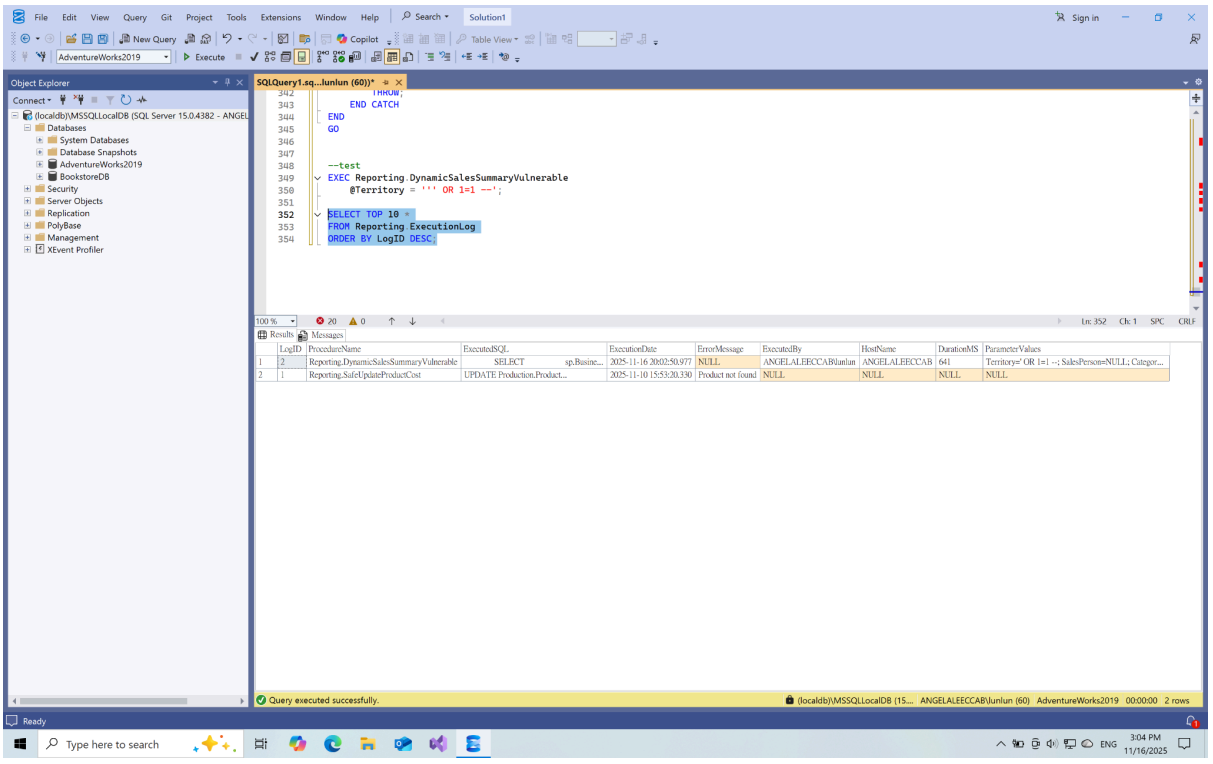


2.3 Vulnerable Procedure

The vulnerable version:

- Uses raw string concatenation
- Runs the SQL through `EXEC(@SQL)`
- Has no validation
- Allows SQL injection on purpose for demonstration



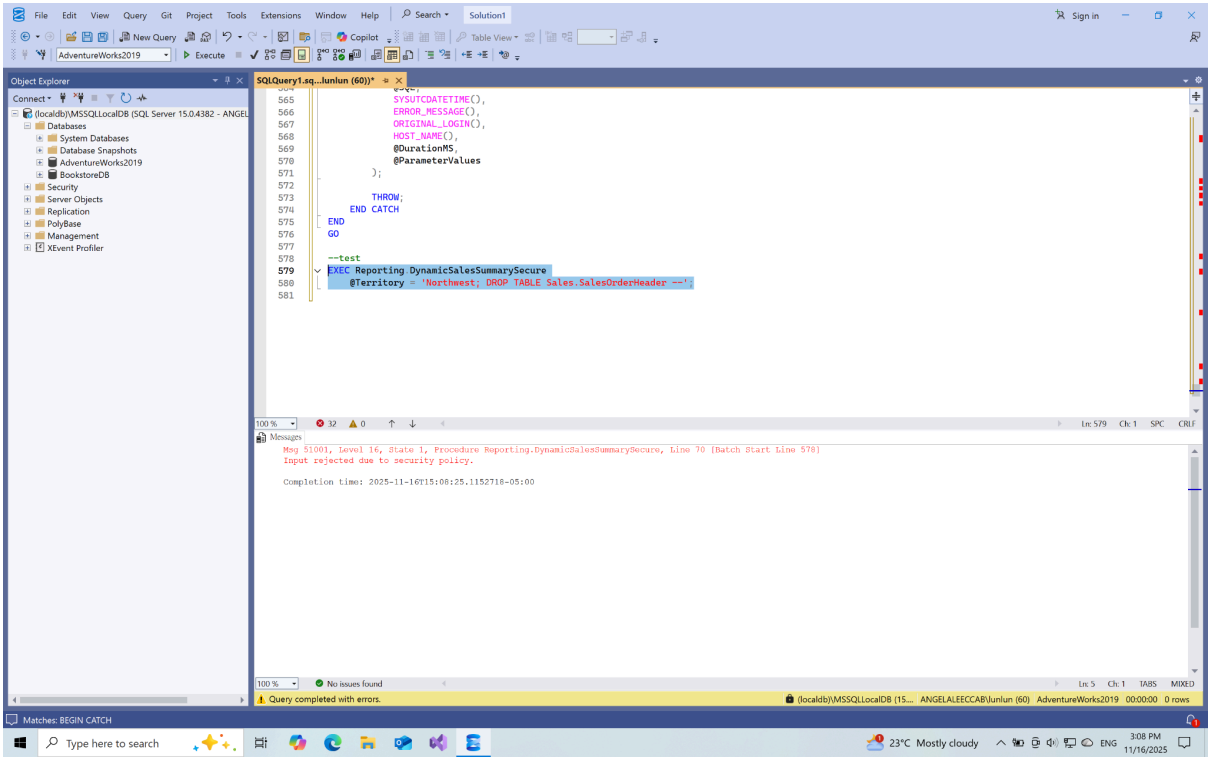


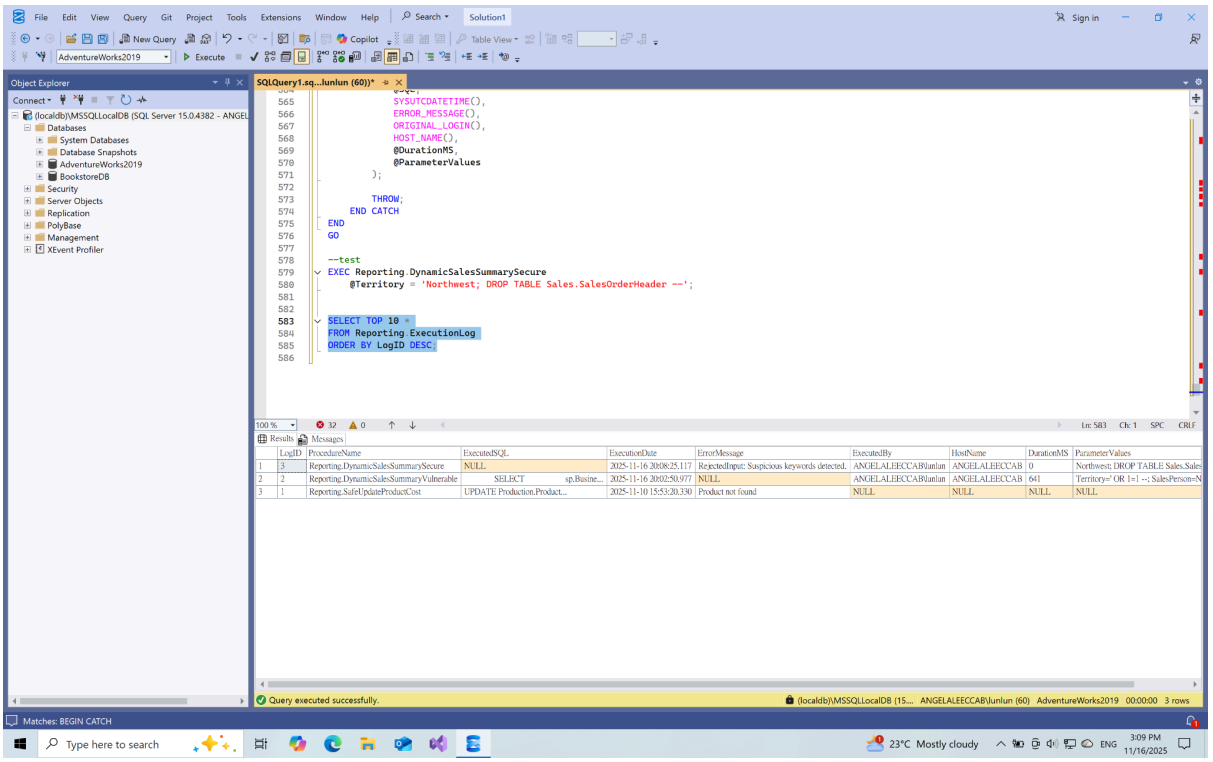
2.4 Input Validation

I added simple validation to the secure procedure.
If a parameter includes keywords such as:

- `;--`
- `DROP`
- `INSERT`
- `EXEC`

The procedure immediately stops, logs the attempt as `RejectedInput`, and returns a safe error message.





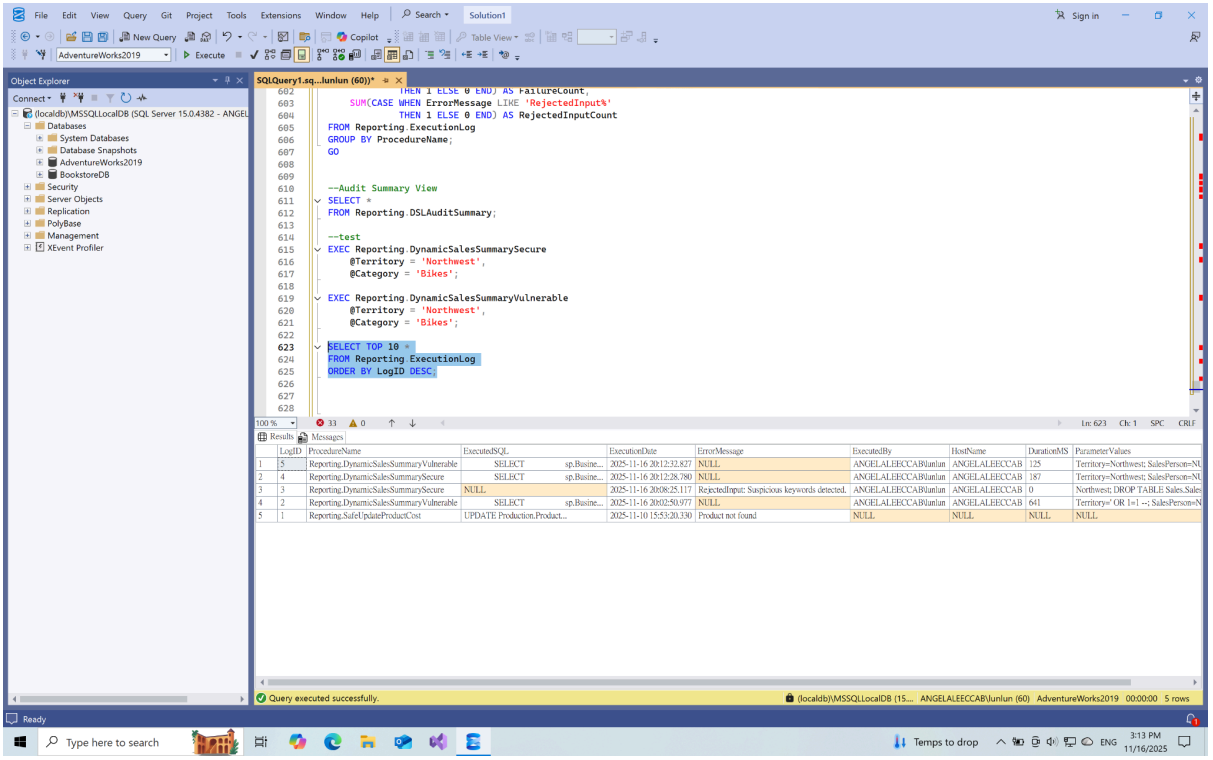
3. Performance Comparison

I compared the execution time of:

- The secure procedure

- The vulnerable procedure

My results showed that the secure version performs better because parameterized SQL can reuse execution plans.
The vulnerable version usually takes longer because string-based SQL must be compiled again each time.

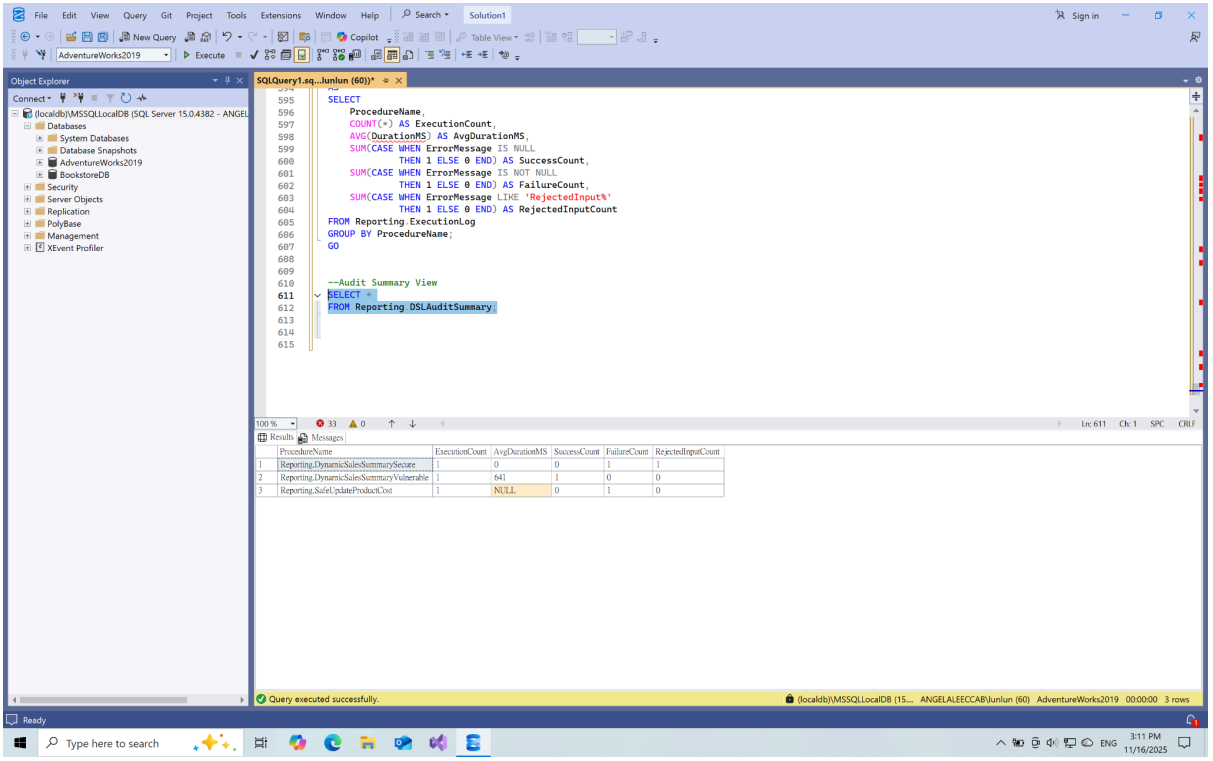


4. Audit Summary

I created a summary view (`Reporting.DSLAuditSummary`) to show:

- How many times each procedure was executed
- Average runtime
- Success and failure counts
- Number of rejected inputs

This makes it easier to analyze overall system behavior.



5. Findings and Recommendations

Findings

- The vulnerable procedure is easy to exploit.
- The secure procedure blocks injection successfully.
- Input validation adds an extra level of protection.
- Logging provides traceability and supports auditing.
- Parameterized SQL gives better performance.

Recommendations

1. Use `sp_executesql` for all dynamic SQL.
2. Add consistent input validation rules.
3. Log all DSL executions.
4. Restrict access to reporting procedures.
5. Review logs regularly for unusual activity.

6. Conclusion

Through this simulation, I learned the practical differences between secure and insecure Dynamic SQL.
By building both versions, I was able to observe:

- How SQL injection works
- How parameterization prevents attacks
- Why logging and validation are important
- Why secure SQL performs better

Overall, this project helped me understand how to build secure and reliable dynamic reporting procedures in SQL Server.