

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

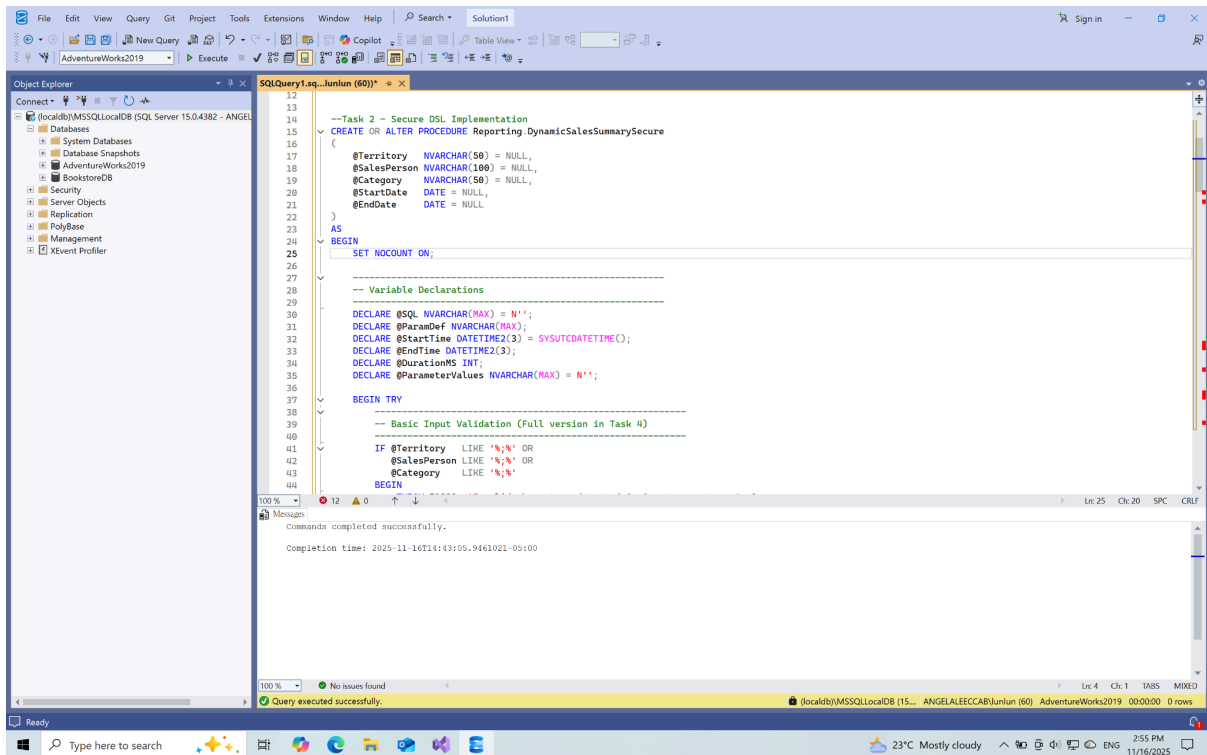
These logs help me review activity, compare results, and identify errors.

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

```

--test
EXEC Reporting.DynamicSalesSummaryVulnerable
@Territory = 'OR 1=1 --';

```

BusinessEntityID	SalesPerson	Territory	Category	OrderDate	TotalDue
283	David Campbell	Northwest	Accessories	2011-05-31 00:00:00.000	16158.6961
282	Joel Saravia	Canada	Accessories	2011-05-31 00:00:00.000	40487.7233
275	Michael Blythe	Northeast	Accessories	2011-05-31 00:00:00.000	4216.0258
278	Garett Vargas	Canada	Accessories	2011-05-31 00:00:00.000	8773.681
281	Shu Ito	Southwest	Accessories	2011-05-31 00:00:00.000	11036.964
281	Shu Ito	Southwest	Accessories	2011-05-31 00:00:00.000	12832.6000
279	Tzvi Reiter	Southeast	Accessories	2011-05-31 00:00:00.000	15524.0686
283	David Campbell	Northwest	Accessories	2011-05-31 00:00:00.000	48204.0662
281	Shu Ito	Southwest	Accessories	2011-05-31 00:00:00.000	43362.4196
275	Michael Blythe	Southeast	Accessories	2011-05-31 00:00:00.000	23126.45
279	Tzvi Reiter	Southeast	Accessories	2011-05-31 00:00:00.000	23242.1865
274	Stephen Jang	Northwest	Accessories	2011-07-01 00:00:00.000	23130.2957
275	Michael Blythe	Northeast	Accessories	2011-07-01 00:00:00.000	7638.8584
282	Joel Saravia	Canada	Accessories	2011-07-01 00:00:00.000	8090.9178
279	Tzvi Reiter	Southeast	Accessories	2011-07-01 00:00:00.000	26346.8927
280	Pamela Ann...	Northwest	Accessories	2011-07-01 00:00:00.000	36066.8893
281	Shu Ito	Central	Accessories	2011-07-01 00:00:00.000	10580.0803
276	Linda Mitchell	Southwest	Accessories	2011-07-01 00:00:00.000	13042.6279
276	Linda Mitchell	Southwest	Accessories	2011-07-01 00:00:00.000	46465.9531
279	Tzvi Reiter	Southeast	Accessories	2011-07-01 00:00:00.000	137343.2...
280	Pamela Ann...	Northwest	Accessories	2011-07-01 00:00:00.000	23232.5397
282	Joel Saravia	Canada	Accessories	2011-07-01 00:00:00.000	24459.114
277	Jillian Carson	Central	Accessories	2011-07-01 00:00:00.000	130416.4...
280	Pamela Ann...	Northwest	Accessories	2011-07-01 00:00:00.000	36556.8816
276	Linda Mitchell	Southwest	Accessories	2011-07-01 00:00:00.000	57683.212
278	Garett Vargas	Canada	Accessories	2011-07-01 00:00:00.000	57391.4284

```

--test
EXEC Reporting.DynamicSalesSummaryVulnerable
@Territory = 'OR 1=1 --';

```

LogID	ProcedureName	ExecutedSQL	ExecutionDate	ErrorMessage	ExecutedBy	HostName	DurationMS	ParameterValues
2	Reporting.DynamicSalesSummaryVulnerable	SELECT	2025-11-16 20:02:50.977	NULL	ANGELA.ECCEAB@junlan	ANGELA.ECCEAB	641	Territory* OR 1=1 --; SalesPerson=NULL; Categor...
1	Reporting.SafeUpdateProductCost	UPDATE ProductionProduct...	2025-11-10 15:53:20.330	Product not found	NULL	NULL	NULL	NULL

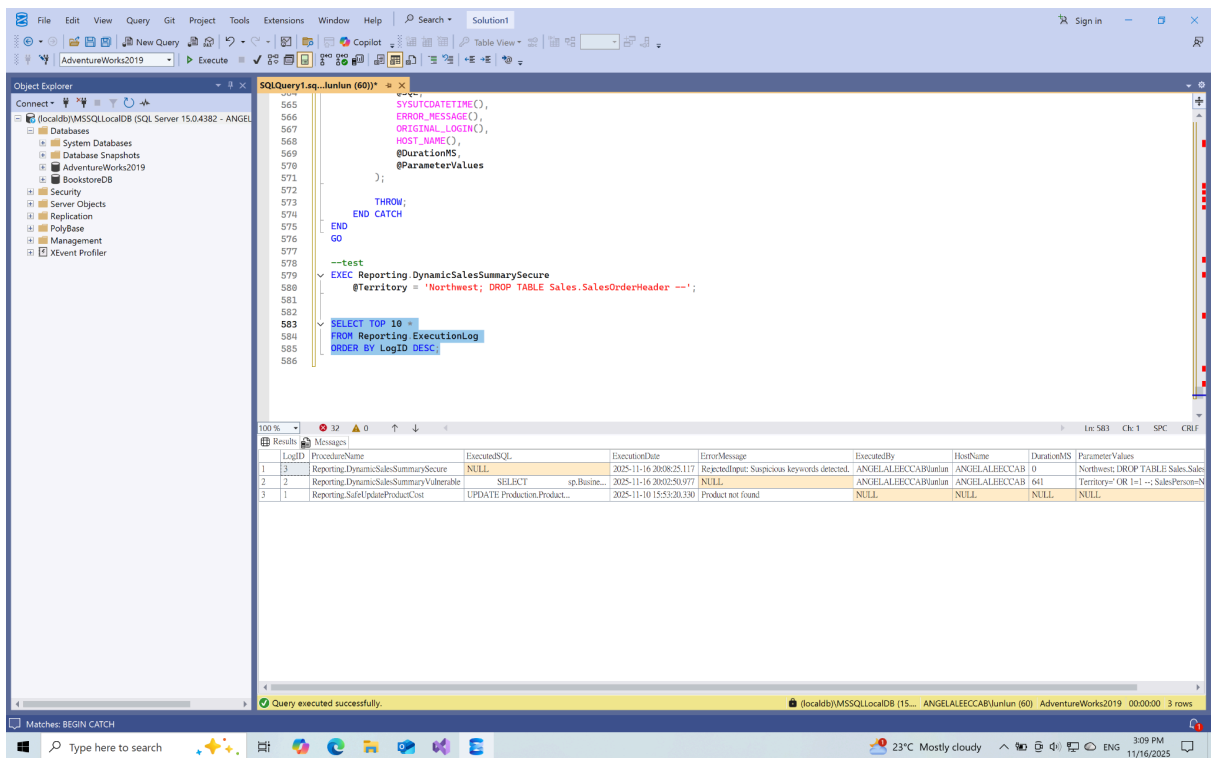
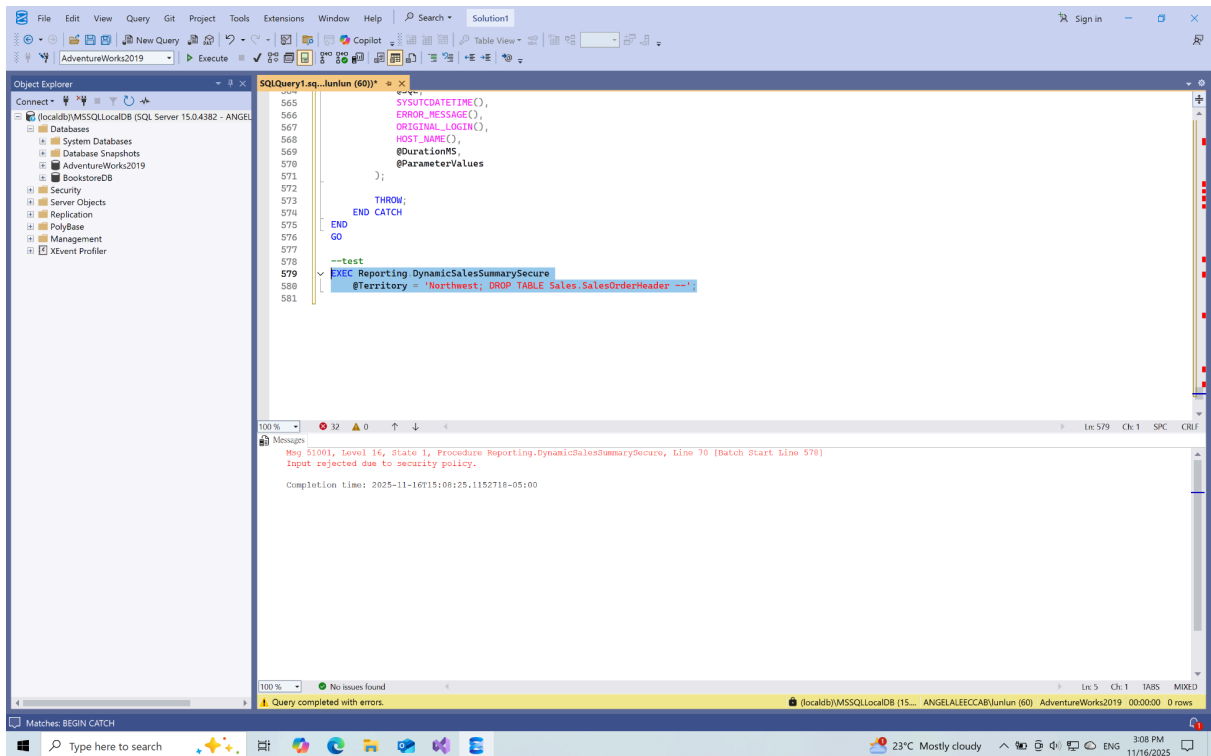
2.4 Input Validation

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.

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the Object Explorer with the AdventureWorks2019 database selected. The right pane shows a query window with the following SQL code:

```

--Audit Summary View
SELECT *
FROM Reporting.DSLAuditSummary;

--test
EXEC Reporting.DynamicSalesSummarySecure
@Territory = 'Northwest',
@Category = 'Bikes';

EXEC Reporting.DynamicSalesSummaryVulnerable
@Territory = 'Northwest',
@Category = 'Bikes';

SELECT TOP 10 *
FROM Reporting.ExecutionLog
ORDER BY LogID DESC;

```

The bottom pane shows the Results tab with the following data:

LogID	ProcedureName	ExecutedSQL	ExecutedBy	HostName	DurationMS	ParameterValues
1	Reporting.DynamicSalesSummaryVulnerable	SELECT	ANGELALECCABJanlan	ANGELALECCAB	125	Territory=Northwest; SalesPerson=...
2	Reporting.DynamicSalesSummarySecure	SELECT	ANGELALECCABJanlan	ANGELALECCAB	187	Territory=Northwest; SalesPerson=...
3	Reporting.DynamicSalesSummarySecure	NULL	ANGELALECCABJanlan	ANGELALECCAB	0	Northwest; DROP TABLE Sales.Sales...
4	Reporting.DynamicSalesSummaryVulnerable	SELECT	ANGELALECCABJanlan	ANGELALECCAB	641	Territory= OR 1=1 --; SalesPerson=...
5	Reporting.SalesUpdateProductCost	UPDATE Production.Product...	NULL	NULL	NULL	NULL

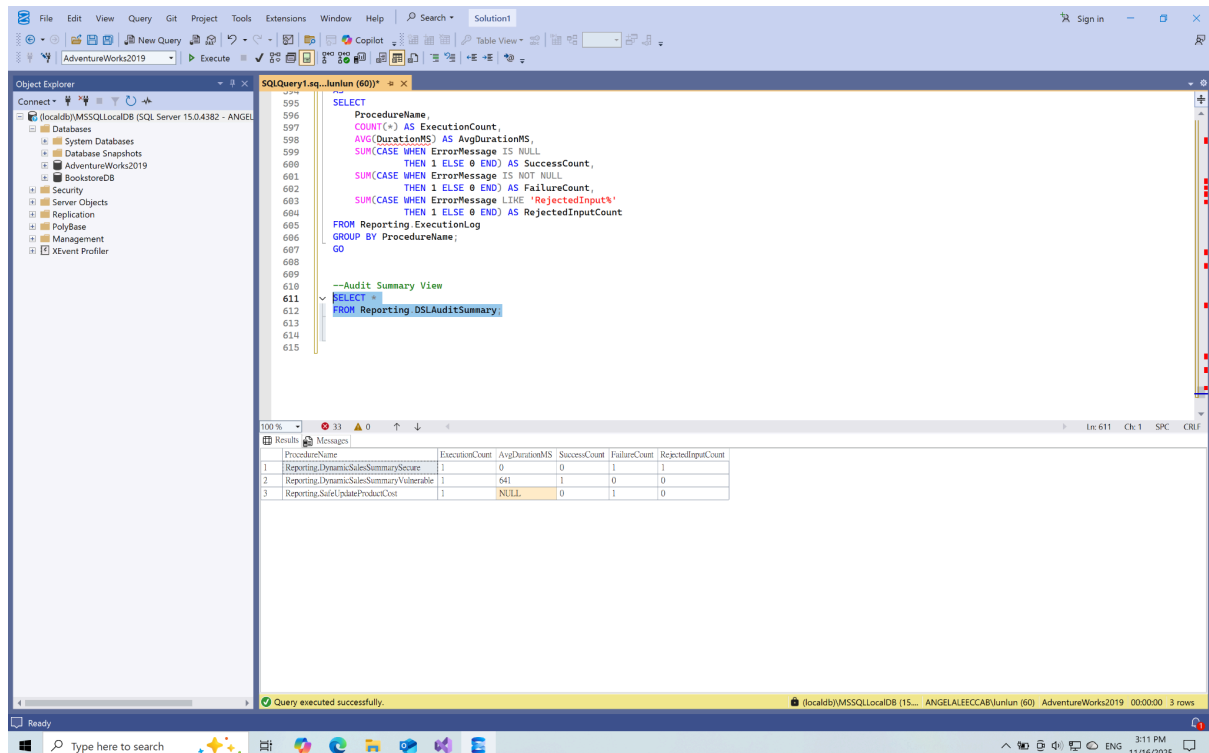
The status bar at the bottom indicates "Query executed successfully." and "5 rows" were returned.

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.