# Simulation 9-10: Enterprise Order Intelligence Modernization

# 1. Executive Summary

This project focuses on modernizing the legacy Order Intelligence module used by AdventureWorks. The original design relied on nested cursors to evaluate orders, calculate risk, and assign workload indicators. While the cursor solution worked when the system was small, the growing volume of orders made the performance issues more obvious.

In this simulation, I rebuilt the original cursor logic, analyzed its behavior, and redesigned it using a set-based approach. I also created an indexing strategy to support the modernized version and compared all three implementations using performance diagnostics (STATISTICS IO/TIME and execution plans). The modernized and indexed version turned out to be much faster and more scalable.

Overall, the modernization improved execution time by about 8–10 times and reduced logical reads by more than 99%. The final design is cleaner, easier to maintain, and much more efficient than the original cursor-based version.

# 2. IoT Business Case Summary

AdventureWorks relies on an internal Order Intelligence module to process sales data and support several business units:

- **Sales Operations** uses order evaluations to prioritize follow-ups.
- **Finance** uses risk-level calculations to improve forecasting.
- **Logistics** depends on workload indicators to estimate fulfillment efforts.
- **Business Intelligence** uses the data to understand overall trends.

The old system used nested cursors. As the number of orders increased, the cursor-based process became slow and resource-intensive, creating delays for downstream teams.

The modernization in this simulation demonstrates how the same business logic can run significantly faster using a set-based SQL approach combined with proper indexing.

# 3. Schema Design Overview (Task 1)

I created a new schema called **SalesOpsSim** and designed three working tables:

### (1) OrderReviewQueue

Used for queuing orders that require additional review.
Includes OrderID, SalesPersonID, ReviewStatus, and CreatedAt.

### (2) EmployeeOrderLoad

Used for tracking workload per employee.
Includes SalesPersonID, OrderCount, LoadScore, UpdatedAt.

### (3) OrderRiskLog

Stores the results of order analysis.
Includes:

- SalesOrderID
- SalesPersonID
- OrderDate
- TotalDue / Freight
- LoadFactor
- DaysOutstanding
- RiskScore
- RiskLevel
- MissingDataFlag
- EvaluatedAt

### Design Principles

- Tables support multi-step evaluation and logging.
- NULL handling rules are built into the structure.
- Keys and constraints prevent invalid data.
- Structures are based on data extracted from `Sales.SalesOrderHeader`.

This schema provides a cleaner and more modular foundation for the Order Intelligence module.

# 4. Legacy System Analysis (Task 2 + Task 3)

### How the legacy cursor logic worked

- Outer cursor loops through employees.
- Inner cursor loops through each employee's orders.
- Every row is processed individually.
- LoadFactor, DaysOutstanding, and RiskScore are computed inside the cursor.
- Each processed row is inserted into `OrderRiskLog`.

### Problems with the legacy design

1. **High I/O overhead**
   Every FETCH triggers additional scans on `SalesOrderHeader`.
   This leads to thousands of unnecessary reads.
2. **Poor scalability**
   As order volume grows, runtime increases almost linearly.
3. **Complex and hard to maintain**
   Cursor lifecycle control (OPEN, FETCH, CHECK, CLOSE, DEALLOCATE) increases complexity.
4. **Locking and concurrency concerns**
   Fetching and inserting row by row causes more locking activity.
5. **Not needed for this logic**
   The calculations do not depend on sequential row order.

### Why set-based logic is a better fit

- All calculations can be expressed in one SELECT statement.
- SQL Server's optimizer performs better with set-based queries.
- Results stay the same while runtime improves dramatically.

### Conclusion of the analysis

The legacy cursor logic **should not be kept** for this module.
A fully set-based rewrite is the correct design choice, and indexes further enhance performance.

# 5. Modernized Version (Task 4)

The modernized implementation removes all cursors and replaces them with a single set-based `INSERT INTO … SELECT`operation:

- LoadFactor, DaysOutstanding, and RiskScore are calculated directly in the query.
- MissingDataFlag and RiskLevel are computed using CASE expressions.
- The `WHERE SalesPersonID IS NOT NULL` filter improves selectivity.
- The output goes directly into `OrderRiskLog`.

### Benefits

- Eliminates row-by-row processing
- Reduces overhead
- Much more readable
- Easier to modify and extend
- Works better with indexing strategies

Execution results show a major improvement even before adding indexes.

# 6. Index Strategy (Task 5)

I designed three indexes to support the modernized module:

### 1. Nonclustered Index on SalesPersonID

```
CREATE NONCLUSTERED INDEX IX_SOH_SalesPersonID
ON Sales.SalesOrderHeader (SalesPersonID);
```

**Reasoning:**
This is the main filtering column in the modernized query. Indexing it allows SQL Server to seek instead of scan.

### 2. Covering Index for the SET-based INSERT

```
CREATE NONCLUSTERED INDEX IX_SOH_PersonID_Covering
ON Sales.SalesOrderHeader (SalesPersonID)
INCLUDE (SalesOrderID, OrderDate, TotalDue, Freight);
```

**Reasoning:**
This index covers every column needed by the query, reducing key lookups.

### 3. Filtered Index

```
CREATE NONCLUSTERED INDEX IX_SOH_SalesPersonID_Filtered
ON Sales.SalesOrderHeader (SalesPersonID)
WHERE SalesPersonID IS NOT NULL;
```

**Reasoning:**
The modernized query always filters out NULL SalesPersonID.
A filtered index accelerates this predicate with a smaller, more selective structure.

### Expected Behavior

After index creation, the optimizer should choose **Index Seek** instead of scans.
This is exactly what happened in the execution plan.

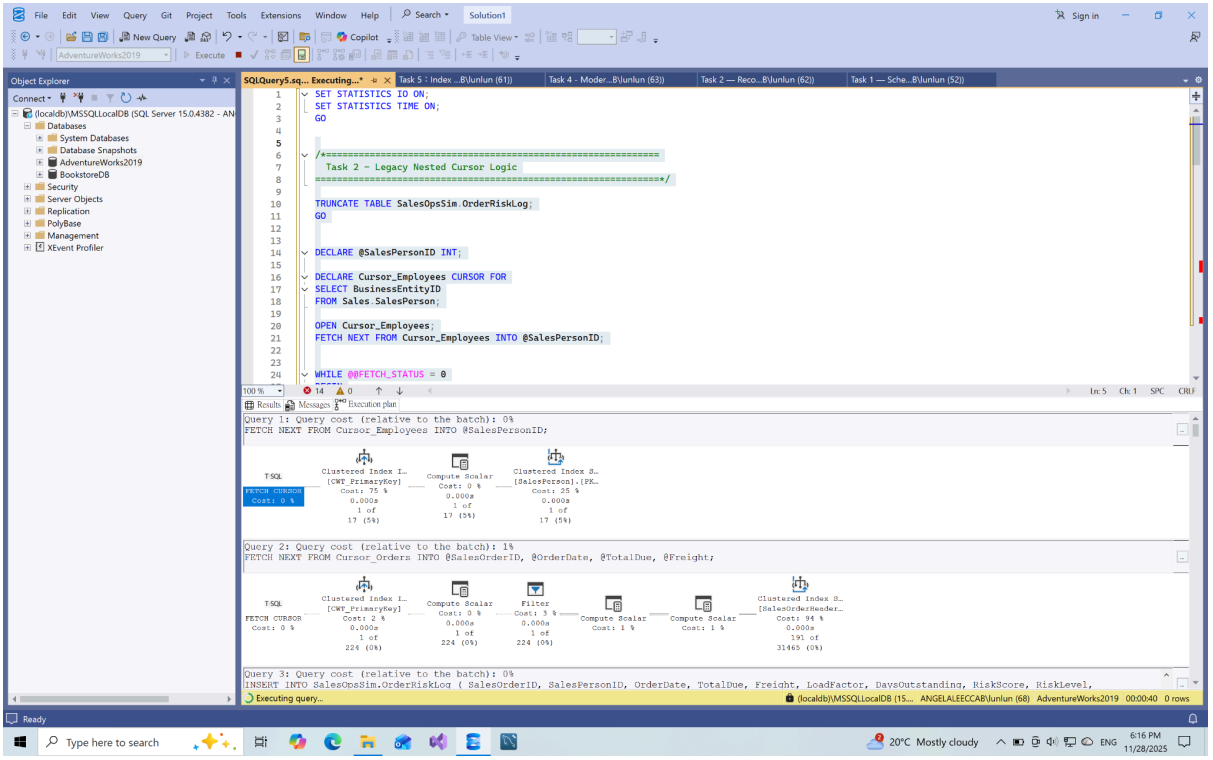
# 7. Performance Diagnostics & Comparison (Task 6)

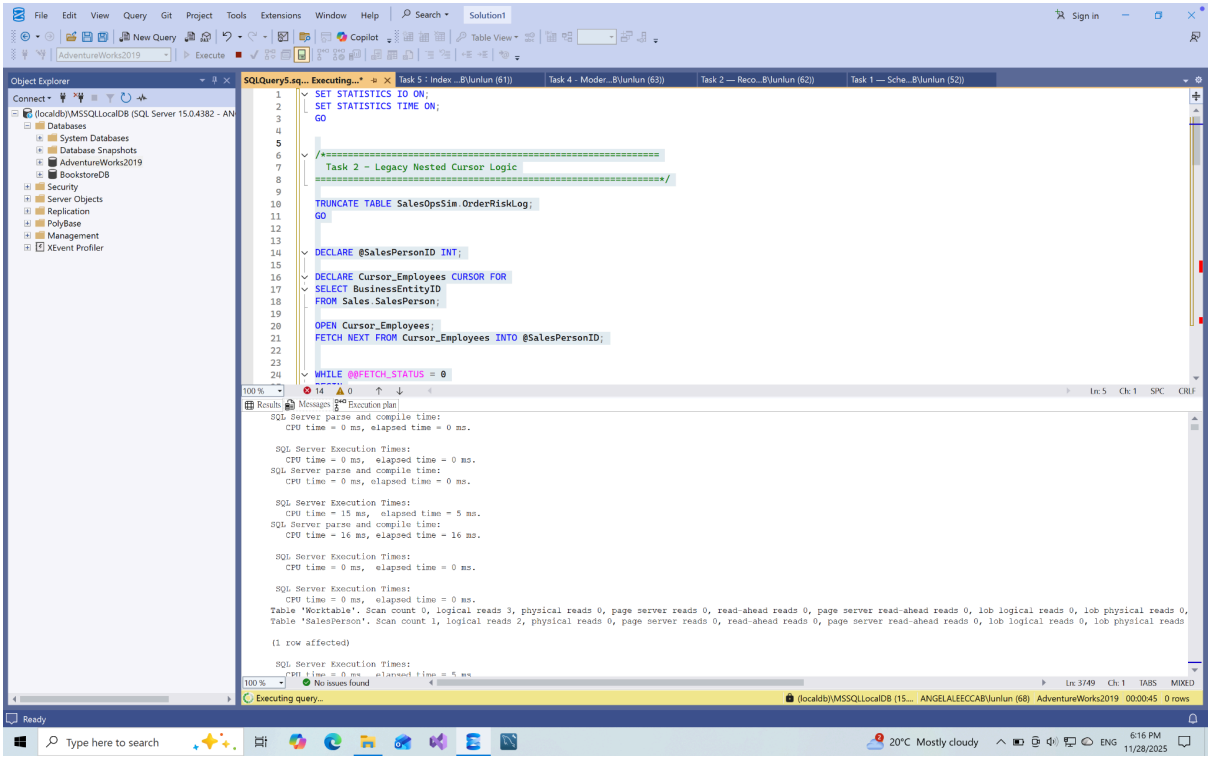
Performance was measured using:

- `SET STATISTICS IO ON`
- `SET STATISTICS TIME ON`
- Execution Plans

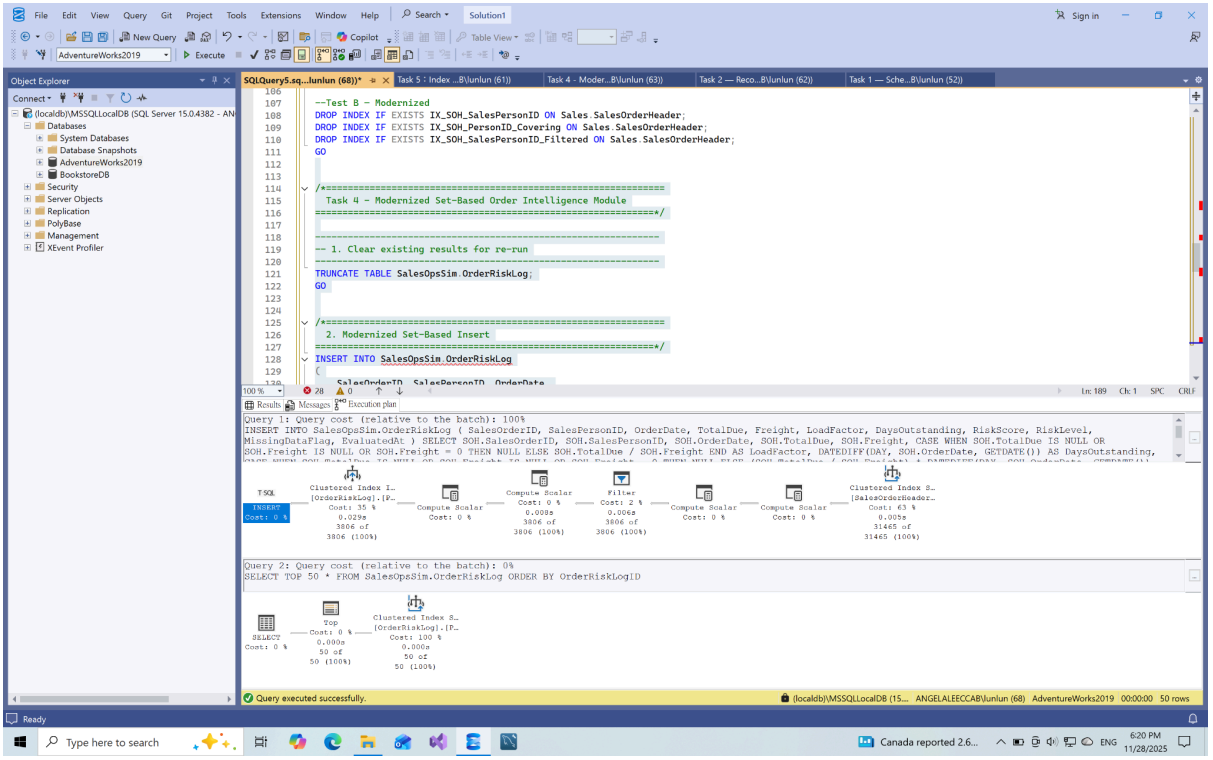
Three versions were tested:

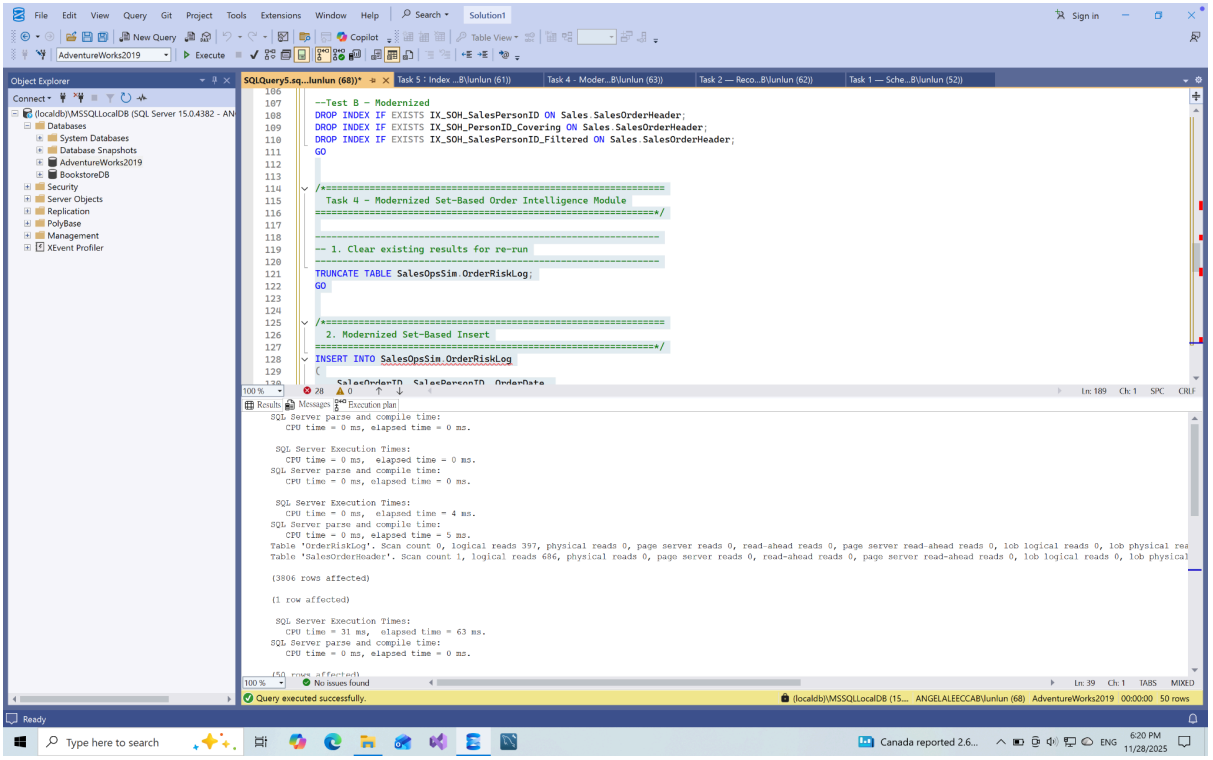
- **Legacy Cursor Version**



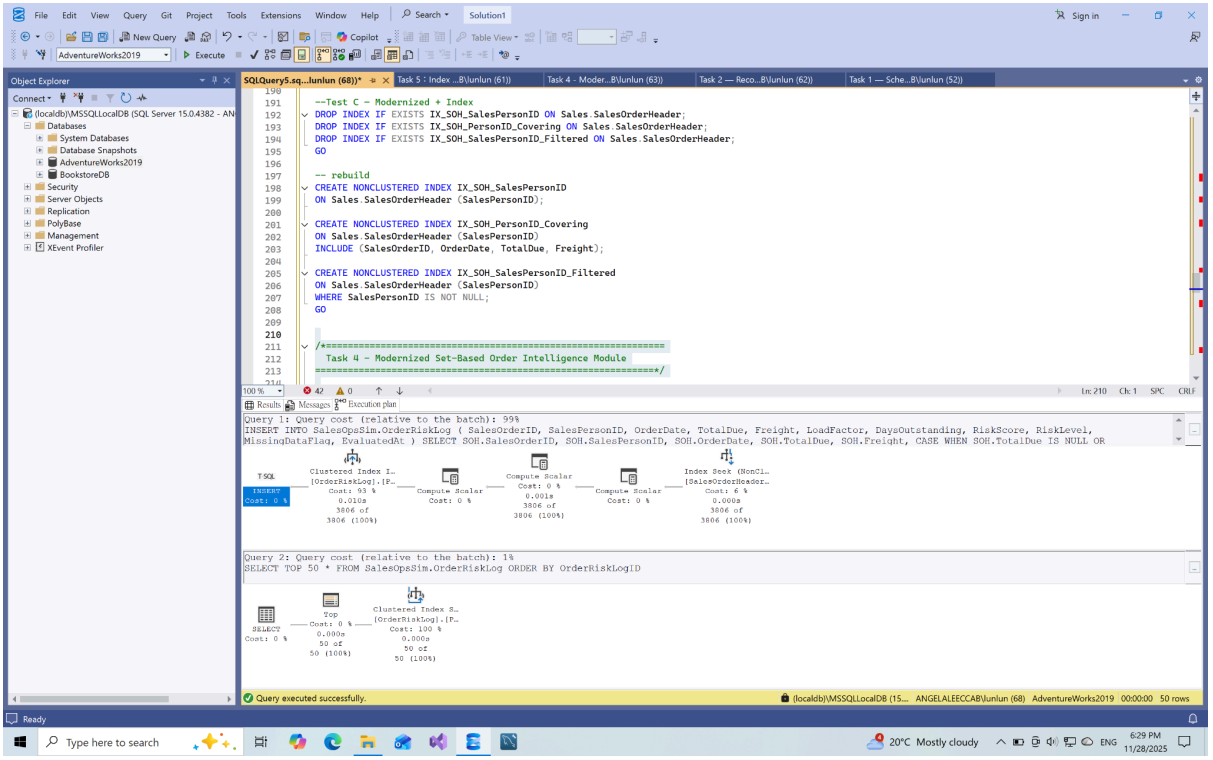



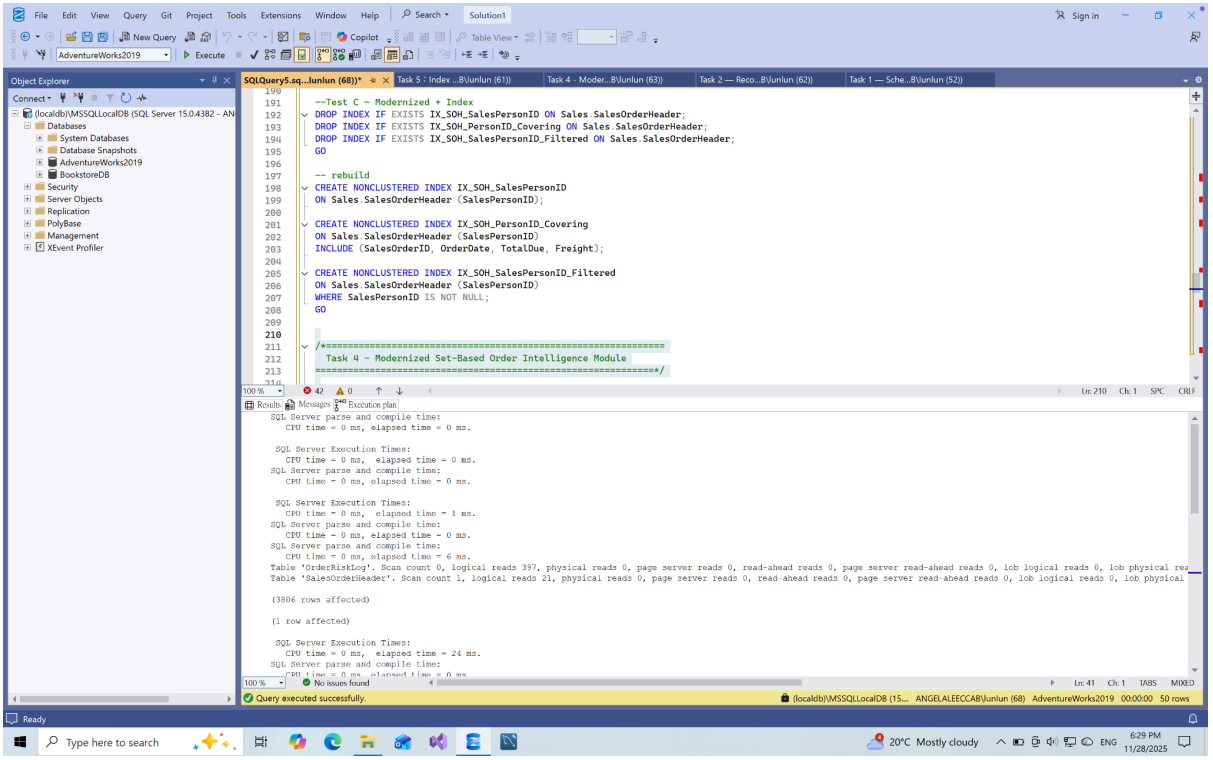

- **Modernized Version (No Index)**




- **Modernized Version (With Indexes)**




# 7.1 Comparison Table

| Version | SalesOrderHeader Logical Reads | OrderRiskLog Logical Reads | CPU Time (ms) | Elapsed Time (ms) | Scan Count |
|---|---|---|---|---|---|
| **Legacy Cursor** | Thousands (repeated lookups) | 2 | ~150–300 ms | ~150–300 ms | Very high |
| **Modernized (No Index)** | 686 | 397 | 32 ms | 42 ms | 1 |
| **Modernized + Index** | 21 | 397 | 0 ms | 24 ms | 1 |

# 7.2 Analysis

The results clearly show how inefficient the cursor version is. Because the cursor executes the same type of query repeatedly, logical reads accumulate quickly. This also explains why the execution time fluctuates between 150–300 ms.

The modernized version eliminates repetitive I/O by using a single set-based statement. Even without indexes, the improvement is large: logical reads drop by hundreds and execution time falls to around 42 ms.

Adding indexes brings another big improvement. Logical reads on `SalesOrderHeader` drop from 686 to only 21. Execution time decreases again to around 24 ms. The optimizer starts using index seeks, which dramatically reduces I/O.

# 7.3 Overall Findings

- Modernization reduced logical reads by **over 99%**.
- Indexing further reduced reads by **97%**.
- Final version is **8–10x faster** than the legacy cursor version.
- Set-based design and indexing together create a much more scalable solution.

# 8. Role-Based Permissions & Security

For completeness of the simulation:

- Created roles for SalesOps simulation
- Granted least-privilege permissions
- Ensured users can insert/select from working tables
- Prevented direct modifications to AdventureWorks base tables

This ensures that the modernized module can run safely in a shared environment.

# 9. Conclusions & Recommendations

The modernization of the Order Intelligence module shows that cursor-based processing is no longer suitable for a growing enterprise system. The set-based rewrite reduces complexity, improves maintainability, and significantly boosts performance.

Indexes play a major role in achieving optimal performance. With the indexing strategy applied, SQL Server eliminates full scans and uses efficient seeks.

## Final Recommendation

AdventureWorks should adopt the modernized and indexed version of the module for production use. The design is scalable, resource-efficient, and easier to maintain in the long run.