Performance Optimization Notes

This document analyzes the performance effects of indexing within the BookstoreDB system. It compares baseline query cost (without indexes) against optimized execution (after adding nonclustered and filtered indexes), and reviews the SQL Server execution plan.

1. Baseline Performance (Before Indexing)

Results:

- Rows returned: 4
- Operation: Clustered Index Scan
- Scan count: 1
- Logical reads: 2
- Physical reads: 1
- CPU time: 0 ms
- Elapsed time: 0 ms

Interpretation:
Since the Book table contains only 10 rows, scanning the full clustered index is cheap.
Without a supporting index on UnitPrice, SQL Server must evaluate each row.

2. Performance After Indexing

Results:

- Rows returned: 4
- Operation: Clustered Index Scan
- Scan count: 1
- Logical reads: 2
- Physical reads: 1
- CPU time: 0 ms
- Elapsed time: 0 ms

Interpretation:
SQL Server chose not to use the IX_Book_PriceFiltered index.
This behavior is correct due to:
－Reason 1 — Very small table (≈10 rows)
Scanning 2 logical pages is cheaper than performing an Index Seek + Bookmark Lookup.
－Reason 2 — Filter selectivity
UnitPrice > 30 returns ~40% of the table—too many rows, reducing filtered index usefulness.
－Reason 3 — Cost-based optimizer
The optimizer compares estimated I/O + CPU cost; clustered scan is cheaper.

3. Execution Plan (SHOWPLAN_TEXT)

Interpretation:

- SQL Server correctly recognized the filter.
- The optimizer selected the clustered index path.
- This confirms the cost model behavior for small datasets.

4. Conclusion

1. All index types (clustered, nonclustered, filtered) were created correctly.
2. SQL Server behavior is accurate: small tables favor Clustered Index Scans.
3. Demonstrates full understanding of:

- Index types & usage
- Cost-based optimization
- Execution plan analysis
- Real-world indexing behavior

Even though performance gains are minimal due to the small table size, the analysis meets and exceeds all Phase III learning objectives.