# Implementing Query Optimization and Performance Comparison

Query optimization is an essential technique for enhancing database performance by reducing unnecessary operations and improving query execution plans. This section demonstrates the impact of query optimization by comparing execution times for a sample query before and after optimization.

## Query Execution Without Optimization (Set 1)

The query retrieves orders for customers located in the USA, filtered by a specific date range. Initially, the query was executed without optimization. The following metrics were recorded:

- \*\*SQL Server Execution Times:\*\*  
 - CPU time: 8 ms  
 - Elapsed time: 10 ms  
- \*\*Total Execution Time:\*\* 00:00:00.291

The query was unoptimized because it selected all columns (`SELECT \*`) and processed unnecessary data, leading to increased execution time and resource usage.

The following code was used to execute the query without optimization:

SET STATISTICS IO ON; -- Enables I/O statistics  
SET STATISTICS TIME ON; -- Enables time measurements  
SELECT \*  
FROM Orders o  
JOIN Customers c ON o.CustomerID = c.CustomerID  
WHERE c.Country = 'USA' AND o.OrderDate >= '2023-01-01';  
SET STATISTICS IO OFF;  
SET STATISTICS TIME OFF;

## Query Execution With Optimization (Set 1)

After optimizing the query to select only required columns and applying proper filters, the query was executed again. The following metrics were recorded:

- \*\*SQL Server Execution Times:\*\*  
 - CPU time: 6 ms  
 - Elapsed time: 5 ms  
- \*\*Total Execution Time:\*\* 00:00:00.101

The query was optimized by explicitly selecting required columns (`SELECT o.OrderID, c.Name, c.Email, o.OrderAmount, o.OrderDate`) instead of using `SELECT \*`. This reduced the data processed and improved performance.

The following code was used to execute the query after optimization:

SET STATISTICS IO ON;  
SET STATISTICS TIME ON;  
SELECT o.OrderID, c.Name, c.Email, o.OrderAmount, o.OrderDate  
FROM Customers c  
INNER JOIN Orders o ON c.CustomerID = o.CustomerID  
WHERE c.Country = 'USA' AND o.OrderDate >= '2023-01-01';  
SET STATISTICS IO OFF;  
SET STATISTICS TIME OFF;

## Performance Comparison (Set 1)

The table below compares the query execution times before and after optimization for the first set of queries:

|  |  |  |
| --- | --- | --- |
| Metric | Without Optimization | With Optimization |
| SQL Server Execution Times (CPU) | 8 ms | 6 ms |
| SQL Server Execution Times (Elapsed) | 10 ms | 5 ms |
| Total Execution Time | 00:00:00.291 | 00:00:00.101 |

## Query Execution Without Optimization (Set 2)

The query retrieves orders for customers located in the USA, filtered by a specific date range. This time, the query was executed using a cross join (Cartesian product), which is inefficient as it processes all combinations of rows from both tables. The following metrics were recorded:

- \*\*SQL Server Execution Times:\*\*  
 - CPU time: 6 ms  
 - Elapsed time: 7 ms  
- \*\*Total Execution Time:\*\* 00:00:00.091

The query was unoptimized because it used a comma-separated syntax for joins, leading to a Cartesian product and processing unnecessary rows.

The following code was used to execute the query without optimization:

SET STATISTICS IO ON; -- Enable I/O statistics  
SET STATISTICS TIME ON; -- Enable time statistics  
SELECT \*  
FROM Orders o, Customers c  
WHERE o.CustomerID = c.CustomerID  
 AND c.Country = 'USA'  
 AND o.OrderDate >= '2023-01-01';  
SET STATISTICS IO OFF;  
SET STATISTICS TIME OFF;

## Query Execution With Optimization (Set 2)

After optimizing the query to use an INNER JOIN with specific column selection, the query was executed again. The following metrics were recorded:

- \*\*SQL Server Execution Times:\*\*  
 - CPU time: 6 ms  
 - Elapsed time: 5 ms  
- \*\*Total Execution Time:\*\* 00:00:00.083

The query was optimized by replacing the Cartesian product with an `INNER JOIN` and selecting only the required columns, which reduced the number of rows processed and improved performance.

The following code was used to execute the query after optimization:

SET STATISTICS IO ON;  
SET STATISTICS TIME ON;  
SELECT o.OrderID, c.Name, c.Email, o.OrderAmount, o.OrderDate  
FROM Customers c  
INNER JOIN Orders o ON c.CustomerID = o.CustomerID  
WHERE c.Country = 'USA' AND o.OrderDate >= '2023-01-01';  
SET STATISTICS IO OFF;  
SET STATISTICS TIME OFF;

## Performance Comparison (Set 2)

The table below compares the query execution times before and after optimization for the second set of queries:

|  |  |  |
| --- | --- | --- |
| Metric | Without Optimization | With Optimization |
| SQL Server Execution Times (CPU) | 6 ms | 6 ms |
| SQL Server Execution Times (Elapsed) | 7 ms | 5 ms |
| Total Execution Time | 00:00:00.091 | 00:00:00.083 |

From the above results, it is evident that query optimization significantly reduces execution time and improves efficiency. By replacing Cartesian joins with INNER JOINs and selecting only relevant columns, database resources are utilized more effectively, enhancing overall performance.