Seminar 5

MS SQL Server
- Performance Tuning -

Query Tuning Methodology

- Identify waits (bottleneck) at the server level
  - I/O latches
  - Log update
  - Blocking
  - Other
- Correlate waits with queues
- Drill down to database/file level
- Drill down to the process level
- Tune problematic queries

DMV - dynamic management views

Identify Waits

- **`sys.dm_os_wait_stats`**:
- Returned table:
  - wait_type
    - Resource waits (locks, latches, network, I/O)
    - Queue waits
    - External waits
  - waiting_tasks_count
  - wait_time_ms
  - max_wait_time_ms
  - signal_wait_time_ms
- reset counters:
  - **`DBCC SQLPERF ('sys.dm_os_wait_stats', CLEAR);`**

Correlate Waits with Queues

- **sys.dm_os_performance_counters**
  - ***object_name*** - Counter category i.e. MSSQL + $ + InstanceName: + Databases (if you have an instance)
  - ***counter_name*** - Counter name relative to the category, which may overlap between various *object_name* values.
  - ***instance_name*** - Instance of the counter which is either a database value or if it is a NULL value than the instance name is related to the overall SQL Server.
  - ***cntr_value*** - The captured or calculated value for the counter.
  - ***cntr_type*** - Counter type defined by Performance Monitor.

Correlate Waits with Queues

- **sys.dm_os_performance_counters**
- > 500 counters: Access Methods, User Settable, Buffer Manager, Broker Statistics, SQL Errors, Latches, Buffer Partition, SQL Statistics, Locks, Buffer Node, Plan Cache, Cursor Manager by Type, Memory Manager, General Statistics, Databases, Catalog Metadata, Broker Activation, Broker/DBM Transport, Transactions, Cursor Manager Total, Exec Statistics, Wait Statistics etc.

- ***cntr_type*** = 65792 → ***cntr_value*** contains actual value
- ***cntr_type*** = 537003264 → ***cntr_value*** contains real-time results need to be divided by a "base" to obtain the actual value. By themselves, they are useless …
  - should divide one value to "base" value to get a ratio, or multiply that result by 100.0 to get a percentage

Correlate Waits with Queues

- **sys.dm_os_performance_counters**

- ***cntr_type*** = 272696576 → ***cntr_value*** contains the base value
  - Counters are time-based
  - Counters are cumulative
  - Use a secondary table to store intermediary values for statistics

- ***cntr_type*** = 1073874176 and ***cntr_type*** = 1073939712
- → poll both value (1073874176) and base value (1073939712)
- poll both values again (let's say, after 15 seconds) ☺
- divide the differences between them to get the result

UnitsPerSec = (cv2 – cv1) / (bv2 – bv1) / 15.0

Drill down to database/file level

- **`sys.dm_io_virtual_file_stats`**
  - Returns I/O statistics for *data files* and *log files*
- Parameters:
  - Database_ID (NULL = all databases), DB_ID function is useful
  - File_ID (NULL = all files), FILE_IDEX function is useful
- Returned tables:
  - Database_ID
  - File_ID
  - Sample_ms - # of milliseconds since the computer was started
  - Num_of_reads - number of physical reads performs
  - Num_of_bytes_read - number of total bytes read
  - Io_stall_read_ms – total time users waited for reads
  - Num_of_writes - number of write performs after the last server/services restarts
  - Num_of_bytes_written - the number of total bytes write
  - Io_stall_write_ms - total time users waited for writes to be completed
  - Io_stall - sum of IO_Stall_Read_ms and IO_Stall_Write_ms
  - File_handle

Drill Down to the Process Level

- A filter on duration/IO will only isolate individual processes (batch/proc/query)

- More important to calculate aggregates on query pattern
  - When using stored procedures, easy to identify pattern
  - When not using stored procedures:
    - Quick and dirty: LEFT(query string, n)
    - Use a parser to identify query pattern

Indexes

- One of the major factors influencing query performance
  - Effect on: filtering, joins, sorting, group by, blocking and dead-lock avoidance, and more
  - Effect on modifications: positive effect on locating the rows; negative effect of cost of modification in index
- Understand indexes and their internals
  - Clustered/nonclustered, single/multicolumn, indexed views and indexes on computed columns, covering scenarios, intersection

Indexes (cont.)

- Depending on environment and the ratio between SELECT queries and data modifications, you should carefully judge whether an additional index maintenance overhead is justified by query performance improvements.
- *Multicolumn indexes* tend to be much more useful than single-column indexes and the query optimizer is more likely to use them to cover the query
- *Indexed views* carry a higher maintenance cost than standard indexes
    - WITH SCHEMABINDING option is mandatory

Fragmentation

- Fragmentation: has a significant effect on query performance
  - *Logical fragmentation*: percent out-of-order pages
  - *Page density*: page population
- Use DBCC SHOWCONTIG to get fragmentation statistics and examine LogicalFragmentation and Avg. Page Density (full)
- Use the *sys.dm_db_index_physical_stats* function, and examine AvgFragmentation
- Rebuild indexes to handle fragmentation

Other statistics

- Update statistics asynchronously
  - ***String summary statistics***: frequency distribution of substrings is maintained for character columns
  - ***Asynchronous auto update statistics*** (default off)
  - **Computed column statistics**

- *sys.dm_exec_query_stats* - performance statistics for cached query plans
  - *total_logical_reads / total_logical_writes* - total number of logical reads/writes performed by executions of a plan since it was compiled.
  - *total_physical_reads* - total number of physical reads performed by executions of this plan since it was compiled.
  - *total_worker_time* - is total amount of CPU time, in microseconds, that was consumed by executions of plan since it was compiled.
  - *total_elapsed_time* - is total elapsed time, in microseconds, for completed executions of the plan.

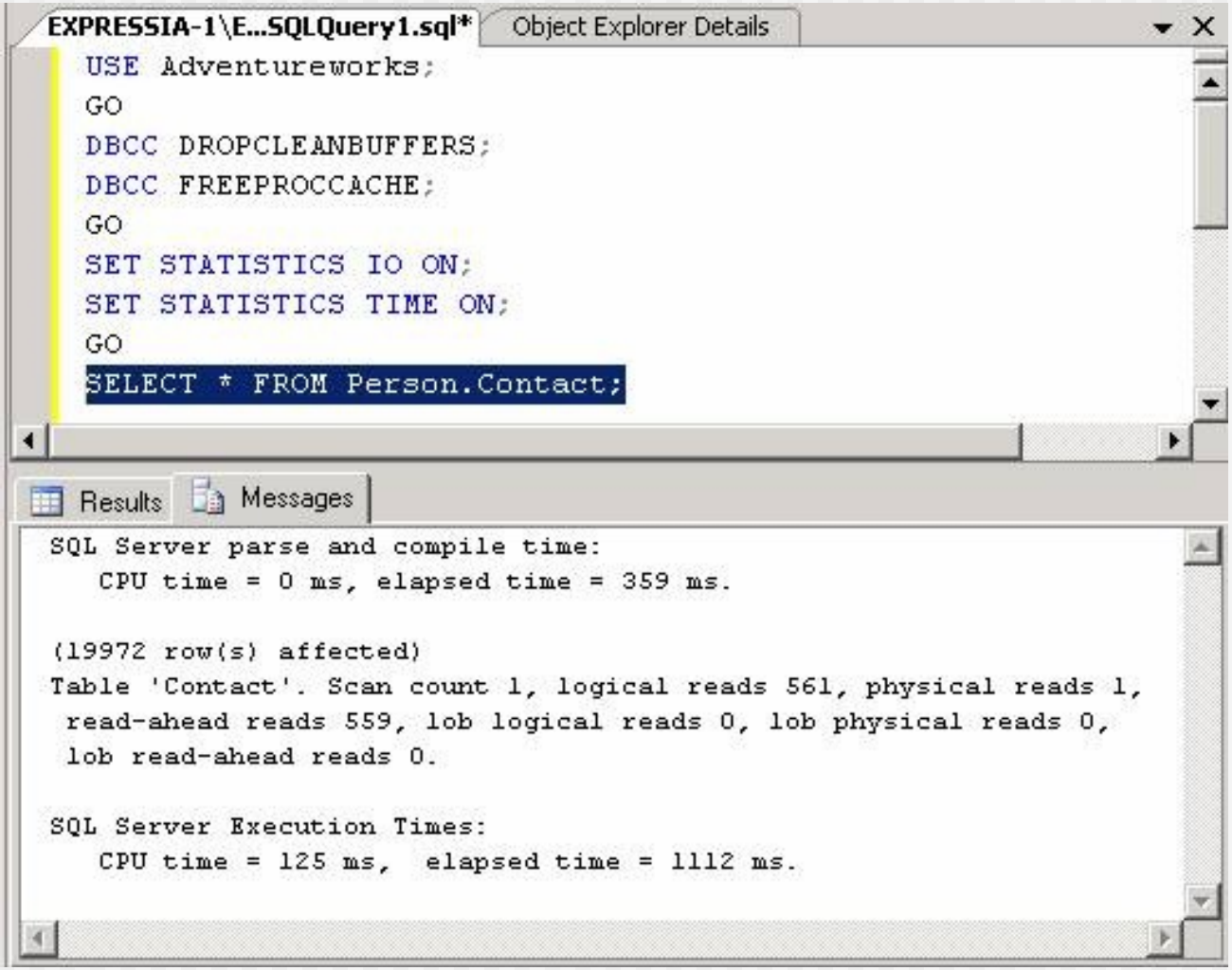
Tools to Analyze Query Performance

- Graphical Execution Plan
- STATISTICS IO: scan count, logical reads, physical reads, read ahead reads
- STATISTICS TIME: duration and net CPU time
- SHOWPLAN_TEXT: estimated plan
- SHOWPLAN_ALL: detailed estimated plan
- STATISTICS PROFILE: detailed actual plan
- SET STATISTICS XML: detailed actual perf info in XML format
- SET SHOWPLAN_XML: detailed estimated perf info in XML format (2005 only)

Query Optimization

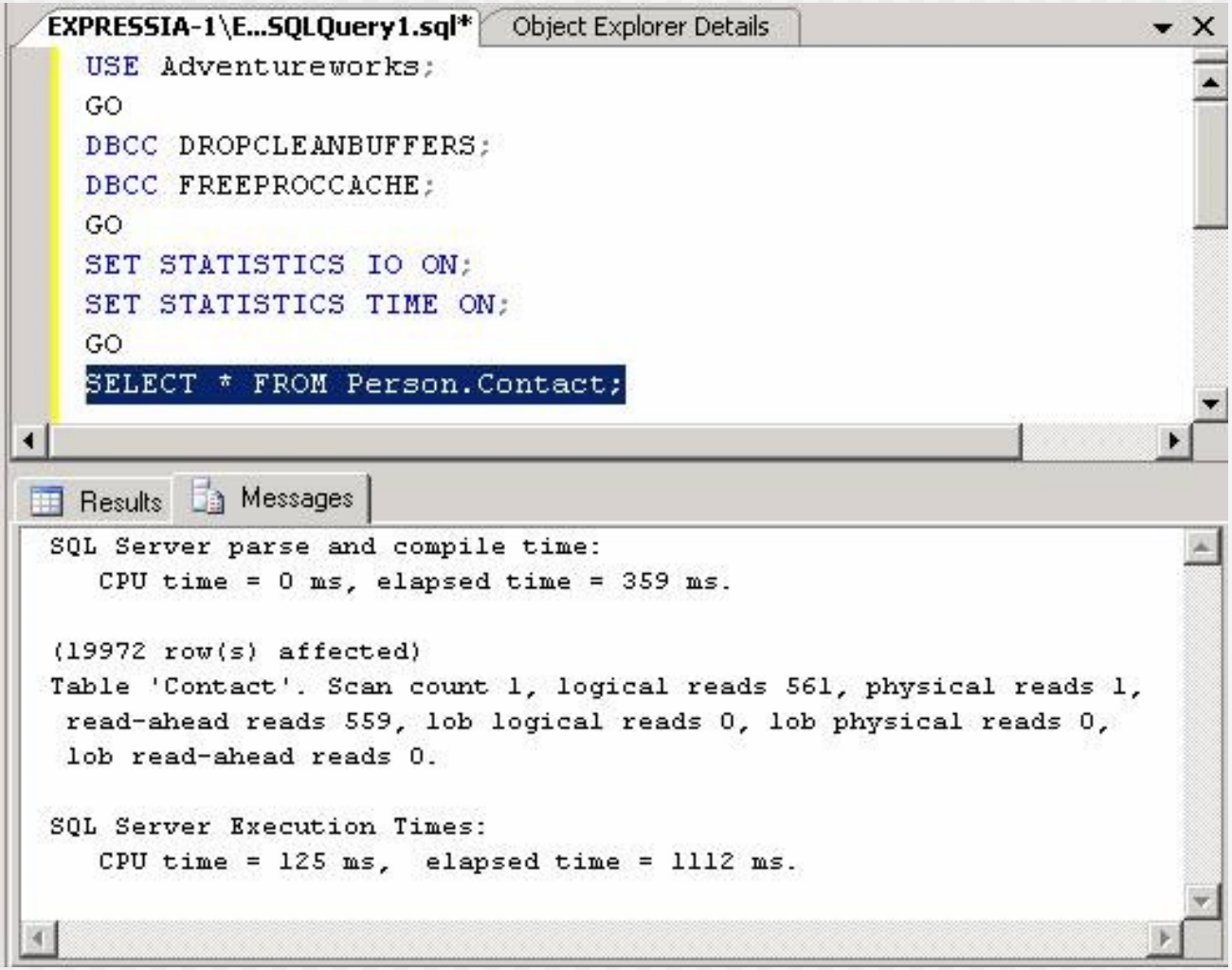
- Evaluation of execution plans
  - Sequence of physical/logical operations
- Optimization factors:
  - Used search predicate
  - Tables involved in joins
  - Join conditions
  - The size of the result
  - List of indexes
- Goal: avoid worst query plans
- SQL Server uses a *cost-based* query optimizer

STATISTICS IO *and* STATISTICS TIME



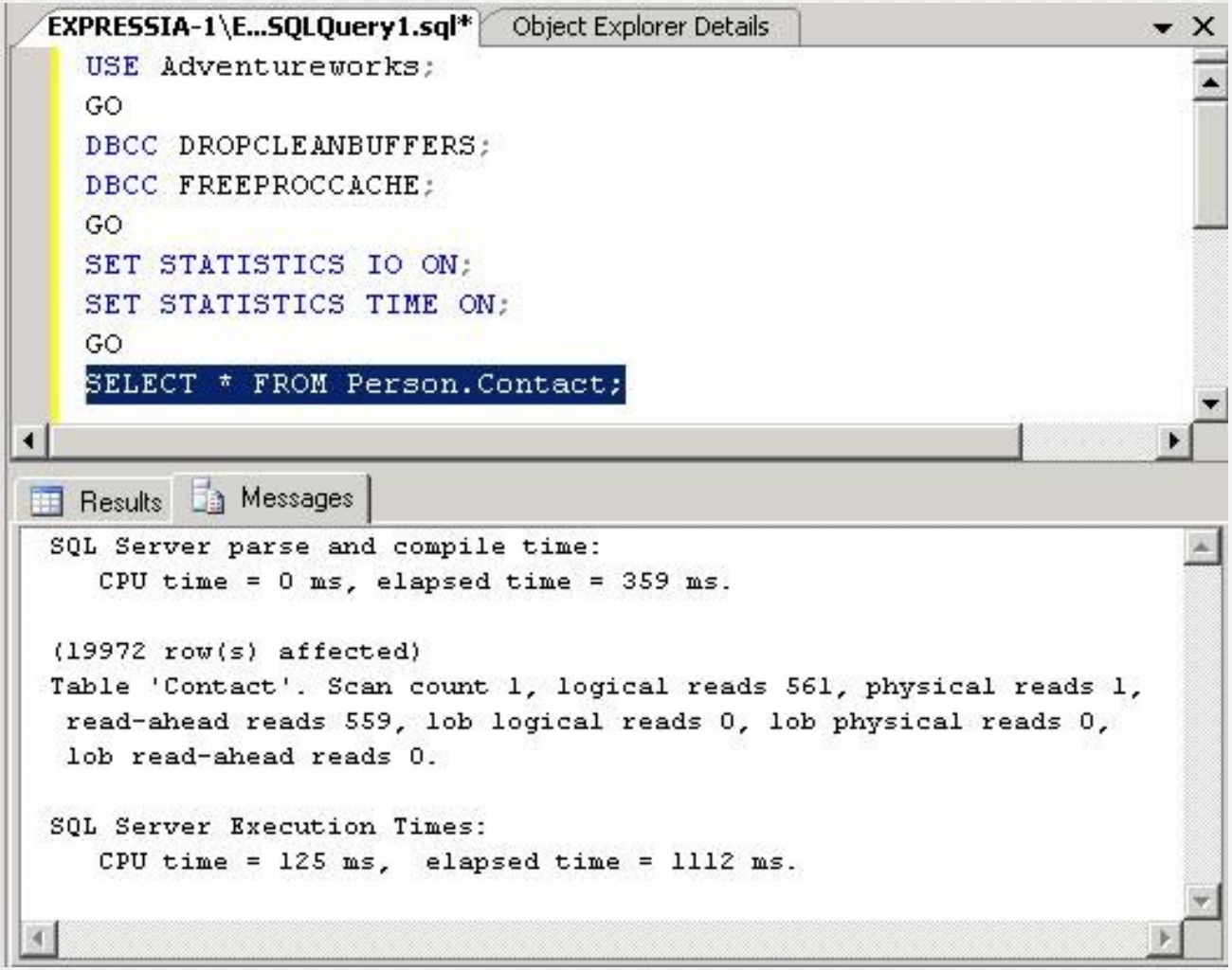
- DBCC DROPCLEANBUFFERS – clears SQL Server data
- DBCC FREEPROCCACHE – clears procedure cache

STATISTICS IO *and* STATISTICS TIME



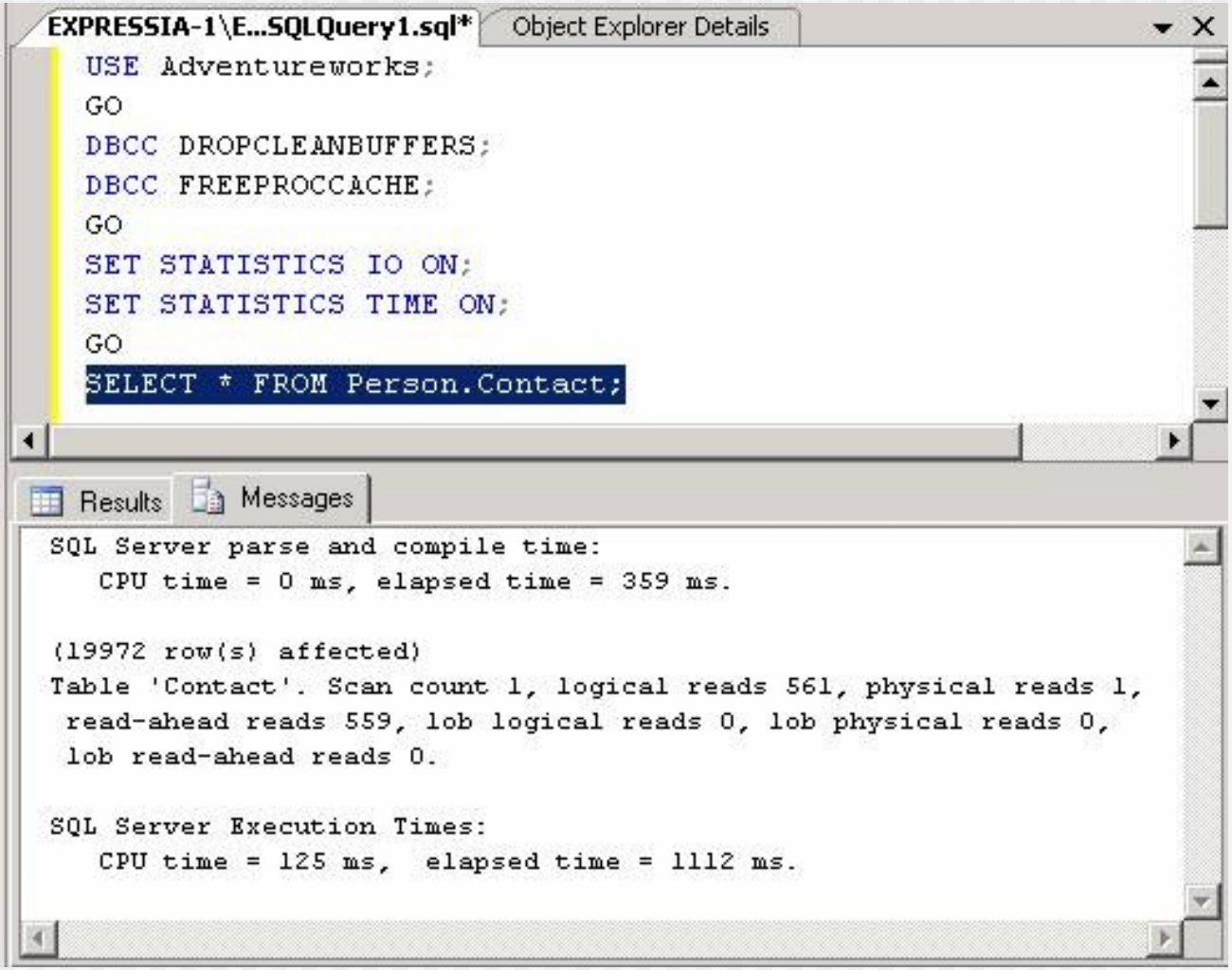
- CPU time – amount of CPU resources used to execute the query
- Elapsed time – how long the query took to execute

STATISTICS IO *and* STATISTICS TIME



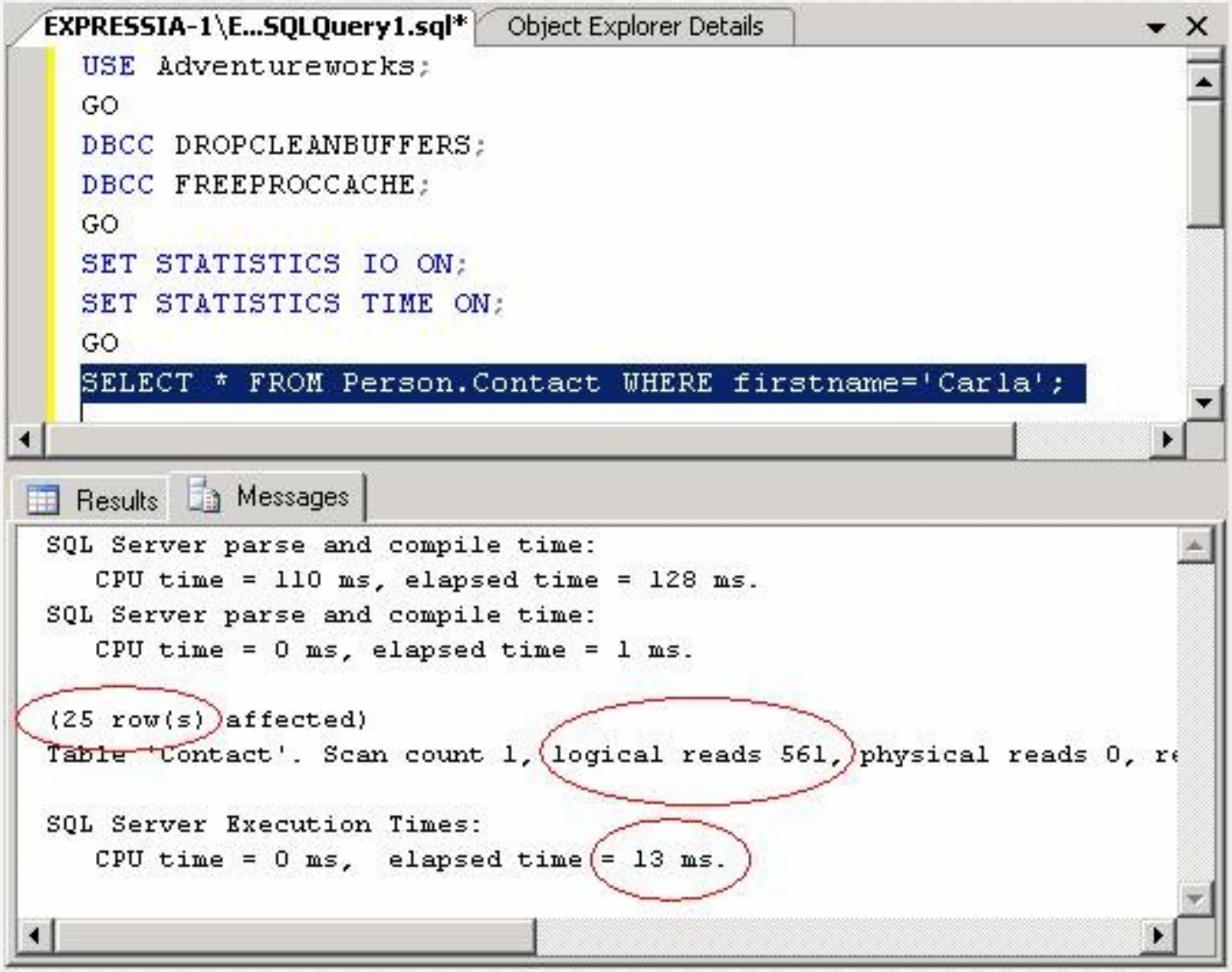
- physical reads --number of pages read from disk
- read-ahead reads - number of pages placed in cache for the query

STATISTICS IO *and* STATISTICS TIME



- Scan count --the number of times that tables have been accessed
- Logical reads - number of pages read from the data cache

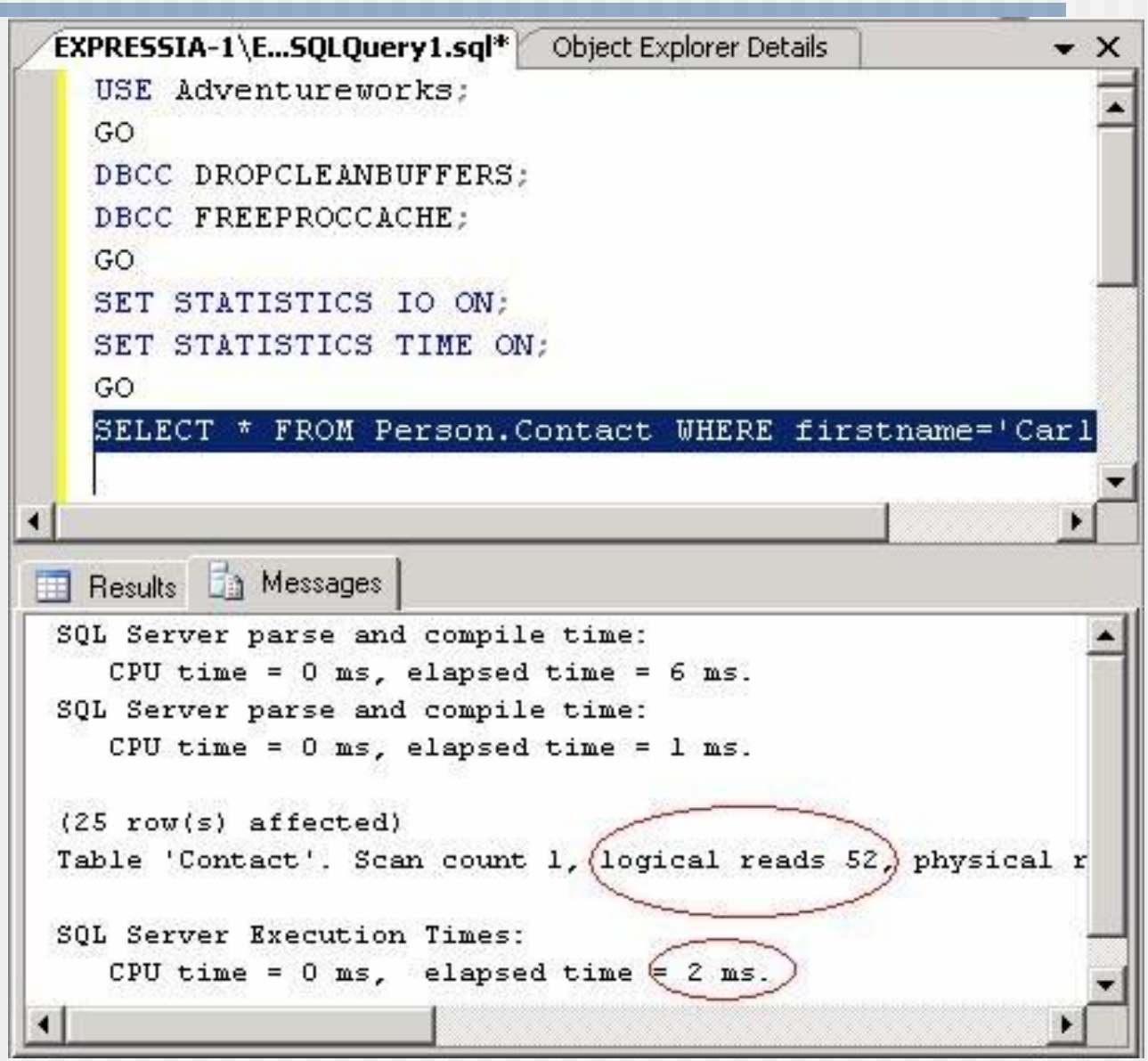
STATISTICS IO *and* STATISTICS TIME



STATISTICS IO *and* STATISTICS TIME

```
USE [AdventureWorks] GO
CREATE NONCLUSTERED INDEX [IDX_firstname]
ON [Person].[Contact]
(
        [FirstName] ASC
)
GO
```

STATISTICS IO *and* STATISTICS TIME



SHOWPLAN_ALL

```
SET SHOWPLAN_ALL ON;
GO
SELECT COUNT(*) cRows
FROM HumanResources.Shift;
GO
SET SHOWPLAN_ALL OFF;
GO
```

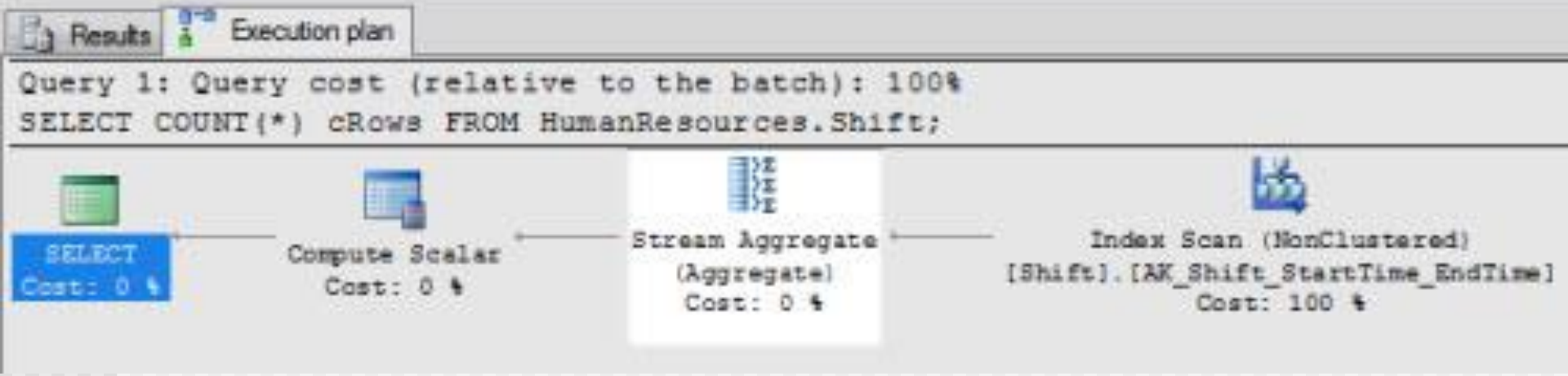
Results

```
StmtText
-----------------------------------------------------------------------------
SELECT COUNT(*) cRows
FROM HumanResources.Shift;
  |--Compute Scalar(DEFINE:([Expr1003]=CONVERT_IMPLICIT(int,[Expr1004],0)))
       |--Stream Aggregate(DEFINE:([Expr1004]=Count(*)))
            |--Index Scan(OBJECT:([master].[HumanResources].[Shift].[AK_Shif

(4 row(s) affected)
```

Graphical execution plan

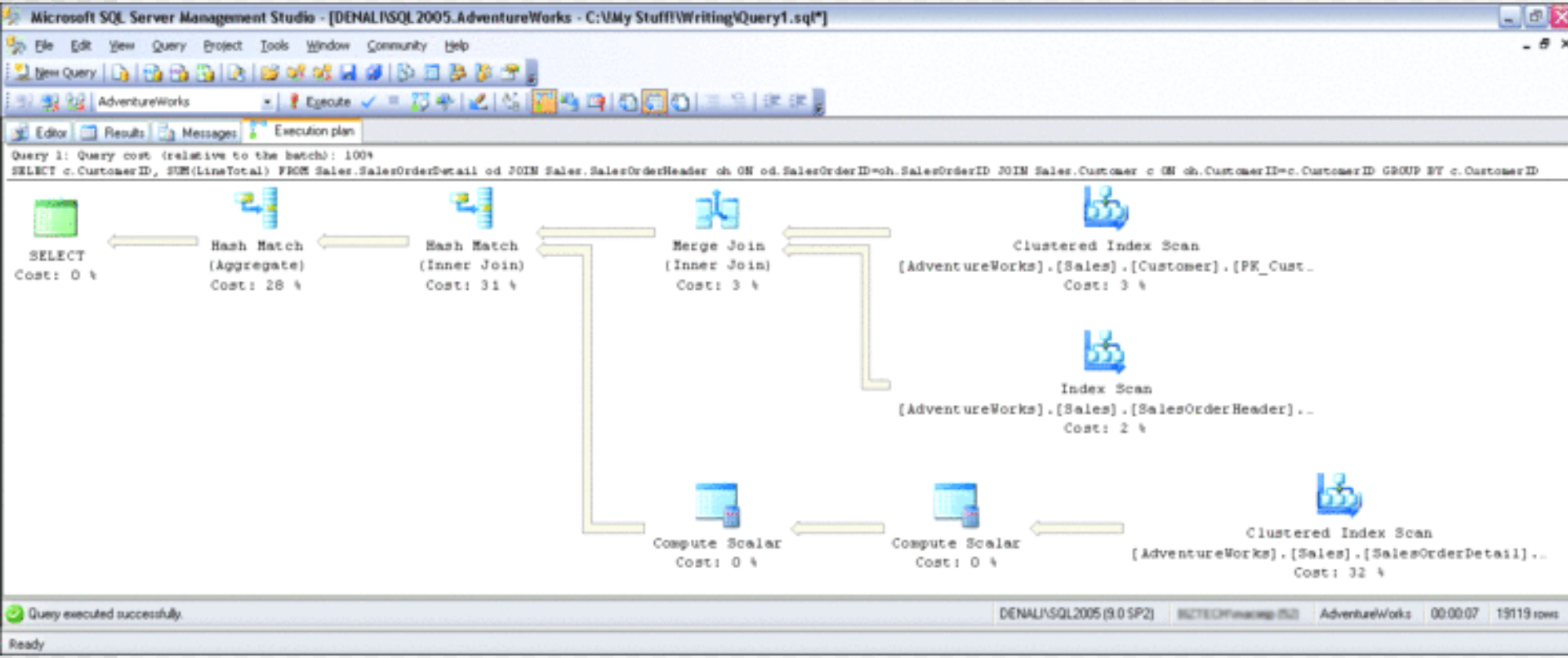
```
USE AdventureWorks
GO
SELECT COUNT(*) cRows
FROM HumanResources.Shift;
GO
```



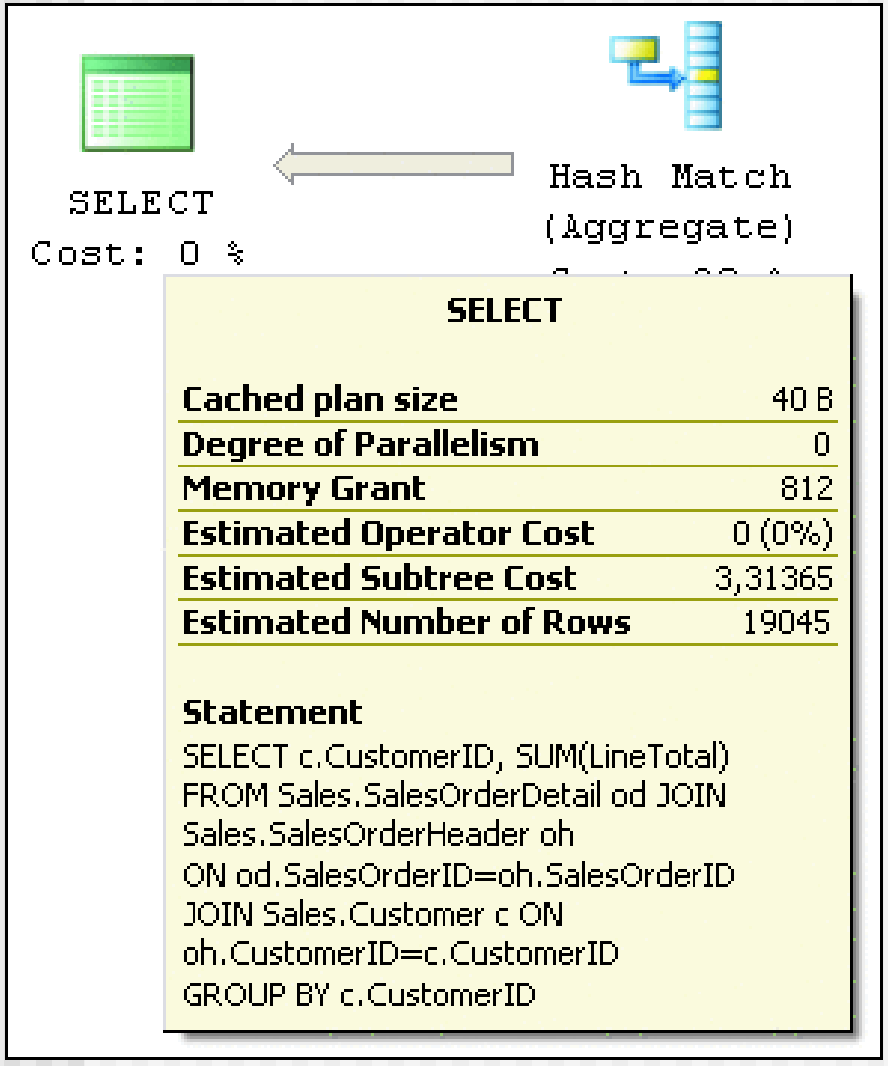
Graphical execution plan (cont.)

```
SELECT c.CustomerID, SUM(LineTotal)
FROM Sales.SalesOrderDetail od
JOIN Sales.SalesOrderHeader oh ON
  od.SalesOrderID=oh.SalesOrderID
JOIN Sales.Customer c ON
 oh.CustomerID=c.CustomerID
GROUP BY c.CustomerID
```

Graphical execution plan (cont.)



Graphical execution plan (cont.)



SELECT

Cached plan size	40 B
Degree of Parallelism	0
Memory Grant	812
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	3,31365
Estimated Number of Rows	19045

Statement

```
SELECT c.CustomerID, SUM(LineTotal)
FROM Sales.SalesOrderDetail od JOIN
Sales.SalesOrderHeader oh
ON od.SalesOrderID=oh.SalesOrderID
JOIN Sales.Customer c ON
oh.CustomerID=c.CustomerID
GROUP BY c.CustomerID
```

Clustered Index Scan

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Number of Rows	121317
Estimated I/O Cost	0,915718
Estimated CPU Cost	0,133606
Estimated Operator Cost	1,04932 (32%)
Estimated Subtree Cost	1,04932
Estimated Number of Rows	121317
Estimated Row Size	29 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	8

Object

[AdventureWorks].[Sales].[SalesOrderDetail].
[PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID]
[od]

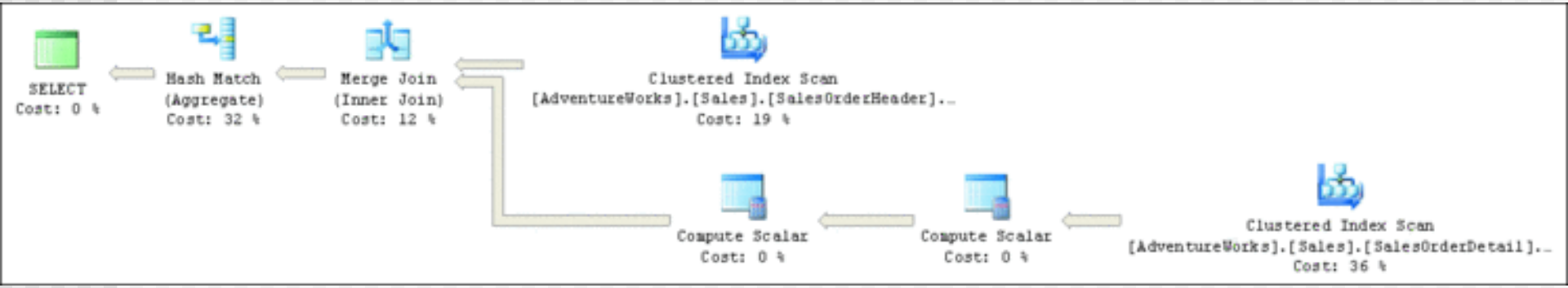
Output List

[AdventureWorks].[Sales].
[SalesOrderDetail].SalesOrderID; [AdventureWorks].
[Sales].[SalesOrderDetail].OrderQty; [AdventureWorks].
[Sales].[SalesOrderDetail].UnitPrice; [AdventureWorks].
[Sales].[SalesOrderDetail].UnitPriceDiscount

Graphical execution plan (cont.)

```
SELECT oh.CustomerID, SUM(LineTotal)
FROM Sales.SalesOrderDetail od
JOIN Sales.SalesOrderHeader oh ON
  od.SalesOrderID=oh.SalesOrderID
GROUP BY oh.CustomerID
```

Graphical execution plan (cont.)



Graphical execution plan (cont.)

```
CREATE INDEX IDX_OrderDetail_OrderID_TotalLine
ON Sales.SalesOrderDetail (SalesOrderID)
INCLUDE (LineTotal)
```