Optimizarea performanței în MS SQL Server

Seminar 5

Optimizarea interogărilor - metodologie

- Identificarea **așteptărilor** (*bottleneck*) la nivel de **server**
  - *I/O latches*
  - *Log* update
  - Blocare
  - Altele
- Corelare așteptări – cozi (*queues*)
- Restrângere la nivel de bază de date/fișier
- Optimizarea interogărilor **problematice**

Latch

- Un *latch* este un **tip special de blocare sistem low-level** care este menținută pe **întreaga durată** a unei **operații fizice** asupra unei **pagini din memorie**, ce are scopul de a **proteja consistența memoriei**
- *Latch*-urile sunt **un mecanism intern al SQL Server** care are scopul de a proteja resursele de memorie **partajate** (cum ar fi **paginile** sau **structurile de date** din **memorie** aflate în *buffer pool*) și de a **coordona** accesul la aceste **resurse**

Latch

- Deoarece *latch*-urile sunt un **mecanism intern** al SQL Server, care **nu** este expus în afara **SQLOS** (*SQL Server Operating System*), ele **nu** pot fi administrate de către utilizatori, spre deosebire de blocările tranzacționale (*locks*), care pot fi administrate cu ajutorul *hint*-urilor NO LOCK
- ***Latch*-urile *I/O*** sunt obținute atunci când se **citesc** sau se **scriu** date **pe disc**
- ***Latch*-urile *buffer*** sunt obținute atunci când se accesează pagini din **memorie**

Identificarea așteptărilor

DMV (Dynamic Management Views) returnează informații despre starea server-ului care pot fi folosite pentru monitorizarea stării server-ului, diagnosticarea problemelor și reglarea performanței

Dynamic management view-ul sistem *sys.dm_os_wait_stats* returnează informații despre toate așteptările întâmpinate de thread-urile care s-au executat

- **wait_type** – numele tipului de așteptare
  - Așteptări resursă (blocări, *latches*, rețea, I/O)
  - Așteptări *queue*
  - Așteptări externe (au loc atunci când un *SQL Server worker* așteaptă terminarea unui eveniment extern, cum ar fi apelul unei proceduri stocate *extended* sau o interogare *linked server*)
- **waiting_tasks_count** – numărul de așteptări pentru tipul de așteptare

Identificarea așteptărilor

- **wait_time_ms** – timpul total de așteptare pentru tipul de așteptare în milisecunde (acest timp include signal_wait_time_ms)
- **max_wait_time_ms** – timpul maxim de așteptare pentru tipul de așteptare
- **signal_wait_time_ms** – diferența dintre momentul în care *waiting thread* a fost semnalat și momentul în care a început să ruleze

Resetarea counter-elor:

```
DBCC SQLPERF ('sys.dm_os_wait_stats', CLEAR);
```

Corelare așteptări - queues

Dynamic management view-ul sistem *sys.dm_os_performance_counters* returnează câte o înregistrare pentru fiecare **performance counter** menținut de server

- **object_name** – categoria de care aparține *counter*-ul
- **counter_name** – numele *counter*-ului relativ la categorie (se poate suprapune pentru diverse valori object_name)
- **instance_name** – numele instanței *counter*-ului (de multe ori conține numele bazei de date)
- **cntr_value** – valoarea înregistrată sau calculată a *counter*-ului
- **cntr_type** – tipul *counter*-ului definit de **Performance Monitor**

Corelare așteptări - queues

Sunt disponibile peste 500 de *counter*-e:

- 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 conține valoarea efectivă
- cntr_type=537003264 → cntr_value conține rezultate în timp real care trebuie împărțite la o "bază" pentru a obține valoarea efectivă (altfel, sunt inutile)
  - valoarea trebuie împărțită la o valoare "bază" pentru a obține un raport, iar rezultatul se poate înmulți cu 100 pentru a-l exprima în procente

Corelare așteptări - queues

- cntr_type=272696576 → cntr_value conține valoarea de bază
  - *Counter*-ele sunt bazate pe timp
  - *Counter*-ele sunt cumulative
  - Se utilizează un tabel secundar pentru stocarea valorilor intermediare pentru statistici
- cntr_type=1073874176 și cntr_type=1073939712
- Se obține atât valoarea (1073874176) cât și valoarea de bază (1073939712)
- Se obțin ambele valori din nou (după 15 secunde)
- Pentru a obține rezultatul vizat, se împart diferențele între ele:

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

Restrângere la nivel de bază de date/fișier

Dynamic management view-ul sistem *sys.dm_io_virtual_file_stats* returnează statistici I/O pentru fișierele de date și loguri

- **Parametri**:
  - database_id (NULL=toate bazele de date), funcția DB_ID este utilă
  - file_id (NULL=toate fișierele), funcția FILE_IDEX este utilă
- **Tabel returnat**:
  - **database_id**
  - **file_id**
  - **sample_ms** – numărul de milisecunde de la pornirea calculatorului
  - **num_of_reads** – numărul de citiri fizice realizate
  - **num_of_bytes_read** – numărul total de octeți citiți

Restrângere la nivel de bază de date/fișier

- **io_stall_read_ms** – timpul total de așteptare al utilizatorilor pentru citiri
- **num_of_writes** – numărul de scrieri efectuate
- **num_of_bytes_written** – numărul total de octeți scriși
- **io_stall_write_ms** – timpul total de așteptare al utilizatorilor pentru finalizarea scrierilor
- **io_stall** – suma io_stall_read_ms și io_stall_write_ms
- **file_handle** – Windows file handle pentru fișier
- **size_on_disk_bytes** – numărul total de octeți folosiți pe disc pentru fișier

- Exemplu:

```
SELECT * FROM sys.dm_io_virtual_file_stats(DB_ID('SGBDIR'),NULL);
```

Indecși

Sunt printre **principalii factori** care influențează **performanța interogărilor**

- **Efect asupra**: filtrării, join-ului, sortării, grupării, evitării blocării și a deadlock-ului, etc.
- **Efect în modificări**: efect **pozitiv** în localizarea înregistrărilor și efect **negativ** al costului modificărilor în index

Înțelegerea indecșilor și a mecanismelor interne ale acestora

- Clustered/nonclustered
- Indecși cu una sau mai multe coloane
- View-uri indexate și indecși pe coloane calculate
- Scenarii de acoperire
- Intersecție

Indecși

- În funcție de mediu și de raportul dintre interogările SELECT și modificările datelor, trebuie să apreciați în ce măsură costul adițional de mentenanță a indecșilor se justifică prin îmbunătățirea performanței interogărilor
- Indecșii cu mai multe coloane tind să fie mult mai utili decât indecșii cu o coloană
- E mai probabil ca *query optimizer*-ul să utilizeze indecși cu mai multe coloane pentru a acoperi o interogare
- View-urile indexate au un cost de întreținere mai ridicat decât indecșii standard
- Opțiunea WITH SCHEMABINDING este obligatorie pentru crearea view-urilor indexate

Unelte pentru analiza performanței interogărilor

- **Plan de execuție grafic**
- **STATISTICS IO**: număr de scanări, citiri logice, citiri fizice, citiri read ahead
- **STATISTICS TIME**: durată și timp CPU net
- **SHOWPLAN_TEXT**: plan estimat
- **SHOWPLAN_ALL**: plan estimat detaliat
- **STATISTICS PROFILE**: plan efectiv detaliat
- **SET STATISTICS XML**: informații detaliate despre performanța efectivă în format XML
- **SET SHOWPLAN_XML**: informații detaliate despre performanța estimată în format XML

Optimizarea interogărilor

Evaluarea planurilor de execuție

- Secvență de operații fizice/logice

Factori de optimizare:

- Predicatul de căutare utilizat
- Tabelele implicate în join
- Condițiile de join
- Dimensiunea rezultatului
- Lista de indecși

Scop: evitarea celor mai slabe planuri pentru interogări

SQL Server utilizează un *query optimizer* bazat pe cost

STATISTICS IO și STATISTICS TIME

```
USE AdventureWorks2014;
GO
DBCC DROPCLEANBUFFERS;
DBCC FREEPROCCACHE;
GO
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO
SELECT * FROM Person.BusinessEntityContact;
```

150 %

Results | Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 249 ms.

(909 rows affected)
Table 'BusinessEntityContact'. Scan count 1, logical reads 8, physical reads 1,
read-ahead reads 8, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 424 ms.
```

- **DBCC DROPCLEANBUFFERS** – elimină toate clean buffers din buffer pool și obiectele columnstore din columnstore object pool
- **DBCC DROPCLEANBUFFERS** se poate folosi pentru a testa interogări utilizând un cold buffer cache fără a da shut down și restart server-ului
- **DBCC FREEPROCCACHE** – șterge toate elementele din *plan cache*

STATISTICS IO și STATISTICS TIME

```
USE AdventureWorks2014;
GO
DBCC DROPCLEANBUFFERS;
DBCC FREEPROCCACHE;
GO
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO
SELECT * FROM Person.BusinessEntityContact;
```

150 %

Results | Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 249 ms.

(909 rows affected)
Table 'BusinessEntityContact'. Scan count 1, logical reads 8, physical reads 1,
read-ahead reads 8, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 424 ms.
```

- **CPU time** – resursele CPU utilizate pentru a executa o interogare
- **Elapsed time** – cât timp a durat execuția interogării

STATISTICS IO și STATISTICS TIME

```
USE AdventureWorks2014;
GO
DBCC DROPCLEANBUFFERS;
DBCC FREEPROCCACHE;
GO
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO
SELECT * FROM Person.BusinessEntityContact;
```

150 %

Results | Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 249 ms.

(909 rows affected)
Table 'BusinessEntityContact'. Scan count 1, logical reads 8, physical reads 1,
read-ahead reads 8, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 424 ms.
```

- **Physical reads** – numărul de pagini citite de pe disc
- **Read-ahead reads** – numărul de pagini plasate în *cache* pentru interogare
- **Scan count** – de câte ori au fost accesate tabelele
- **Logical reads** – numărul de pagini citite din *data cache*

STATISTICS IO și STATISTICS TIME

```
USE AdventureWorks2014;
GO
DBCC DROPCLEANBUFFERS;
DBCC FREEPROCCACHE;
GO
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO
SELECT ReorderPoint FROM Production.Product WHERE ReorderPoint>375;
```

150 %

Results | Messages

```
SQL Server parse and compile time:
   CPU time = 16 ms, elapsed time = 376 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

(181 rows affected)
Table 'Product'. Scan count 1, logical reads 15, physical reads 1, read-ahead re

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 30 ms
```

STATISTICS IO și STATISTICS TIME

```
--Se definește un index pentru a optimiza interogarea

USE [AdventureWorks2014];

GO

CREATE NONCLUSTERED INDEX [IX_Product_ReorderPoint_ASC]

ON [Production].[Product]

(ReorderPoint ASC);

GO
```

STATISTICS IO și STATISTICS TIME

```
USE AdventureWorks2014;
GO
DBCC DROPCLEANBUFFERS;
DBCC FREEPROCCACHE;
GO
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO
SELECT ReorderPoint FROM Production.Product WHERE ReorderPoint>375;
```

150 %

Results | Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 105 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

(181 rows affected)
Table 'Product'. Scan count 1, logical reads 2, physical reads 1, read-ahead rea

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 9 ms.
```

SHOWPLAN_ALL

- Dacă **SHOWPLAN_ALL** este setat pe ON, Microsoft SQL Server nu execută instrucțiunile Transact-SQL, ci returnează informații detaliate despre cum sunt executate instrucțiunile și oferă estimări ale cerințelor de resurse pentru instrucțiuni
- **SHOWPLAN_ALL** returnează informații sub forma unui set de înregistrări care formează un *hierarchical tree* ce reprezintă pașii efectuați de *SQL Server query processor* pe măsură ce execută fiecare instrucțiune
- Fiecare instrucțiune reflectată în *output* conține o singură înregistrare cu textul instrucțiunii, urmată de mai multe înregistrări cu detaliile pașilor de execuție
- Sintaxa:

```
SET SHOWPLAN_ALL { ON | OFF }
```

SHOWPLAN_ALL

- Exemplu:

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

100 %

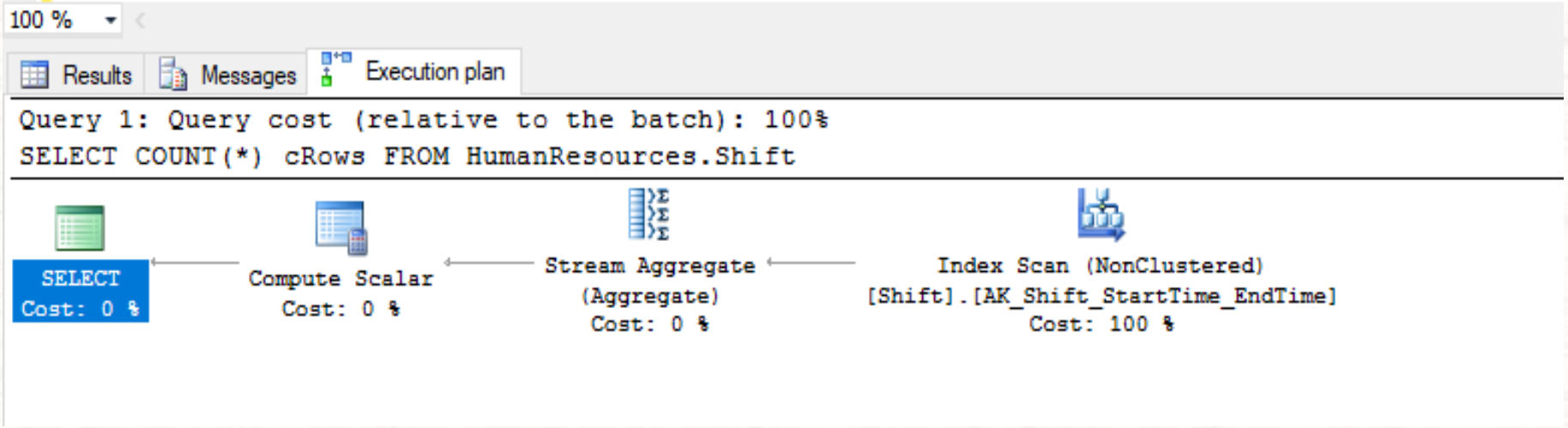
Results | Messages

	StmtText
1	SELECT COUNT(*) cRows FROM HumanResources.Shift;
2	|--Compute Scalar(DEFINE:([Expr1002]=CONVERT_IMPLICIT(int,[Expr1003],0)))
3	|--Stream Aggregate(DEFINE:([Expr1003]=Count(*)))
4	|--Index Scan(OBJECT:([AdventureWorks2012].[HumanResources].[Shift].[AK_Shift_StartTime_EndTime]))

Plan de execuție grafic

- Exemplu:

```
USE AdventureWorks2014
GO
SELECT COUNT(*) cRows FROM HumanResources.Shift;
GO
```

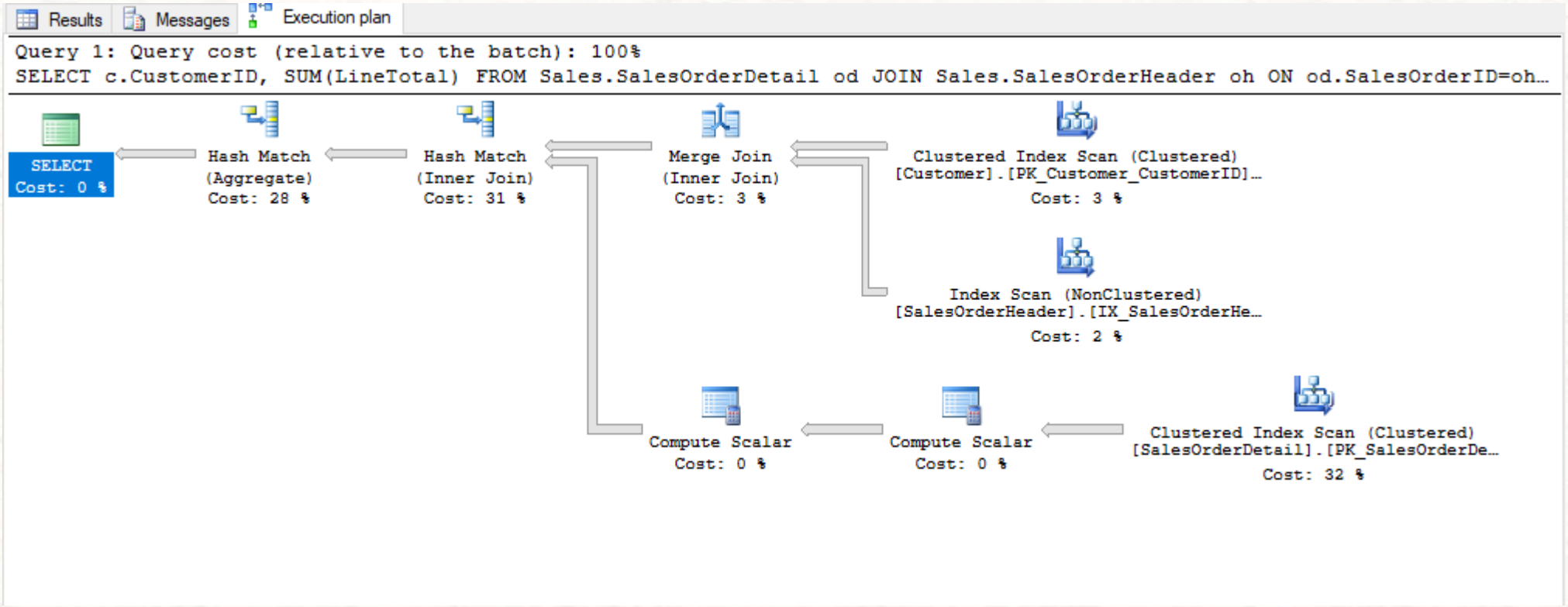


Plan de execuție grafic

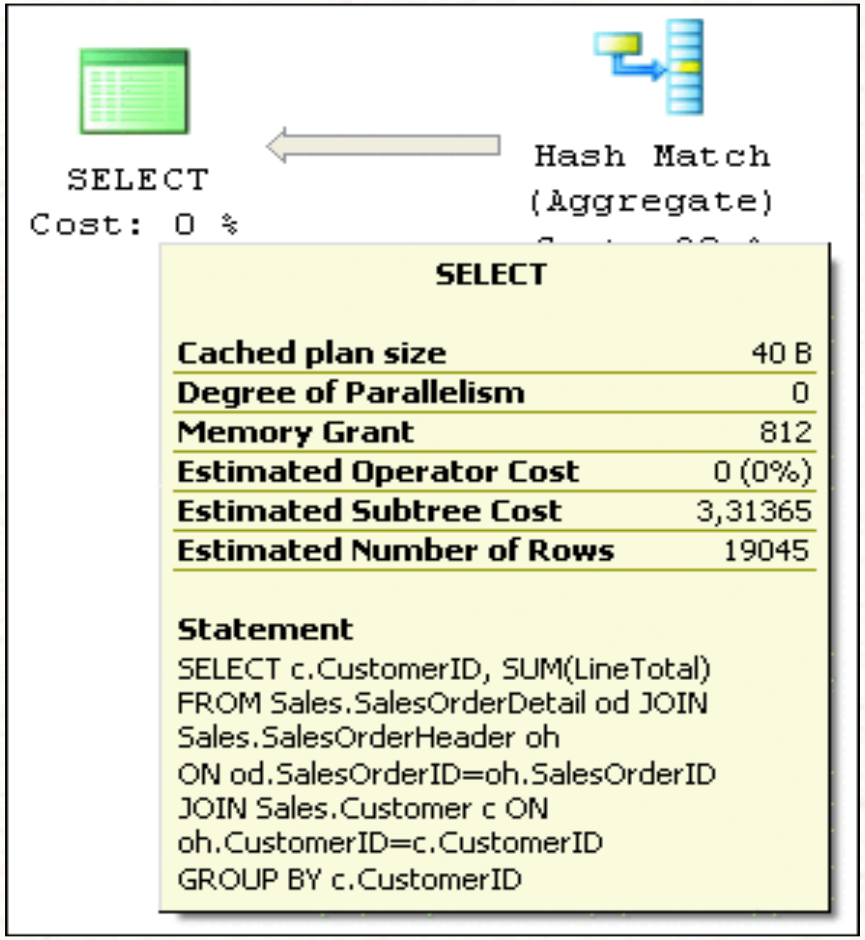
- Exemplu:

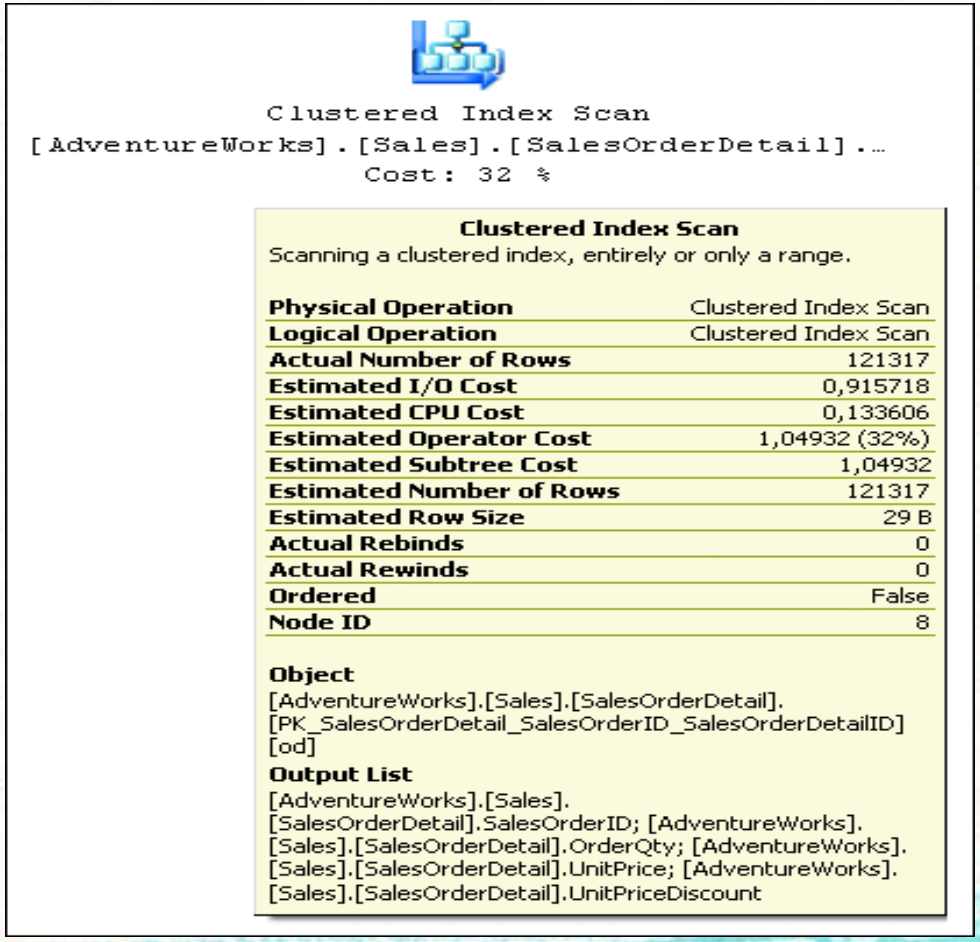
```
SELECT c.CustomerID, SUM(LineTotal)
FROM Sales.SalesOrderDetail od
JOIN Sales.SalesOrderHeader oh ON
od.SalesOrderID=oh.SalesOrderID
JOIN Sales.Customer c ON
oh.CustomerID=c.CustomerID
GROUP BY c.CustomerID;
```

Plan de execuție grafic



Plan de execuție grafic



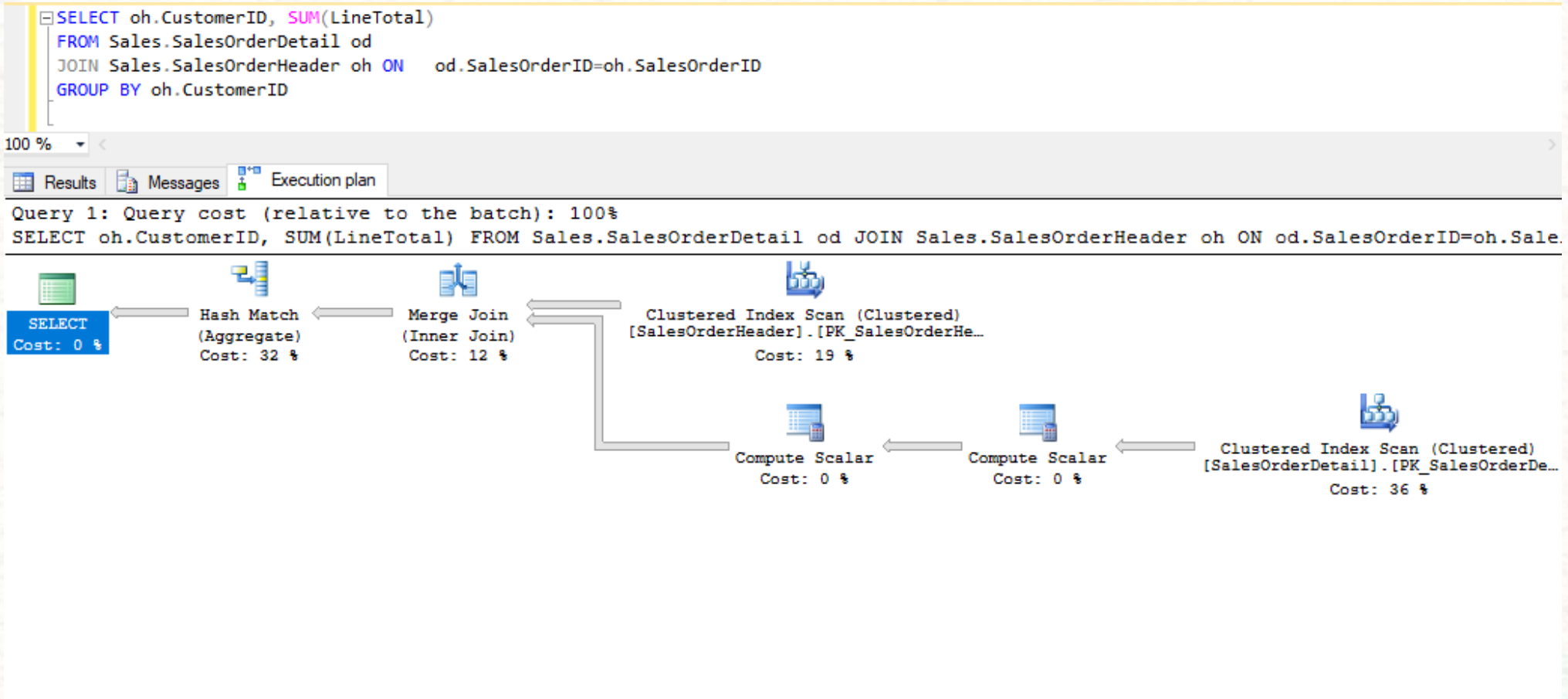


Plan de execuție grafic

- Exemplu:

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

Plan de execuție grafic

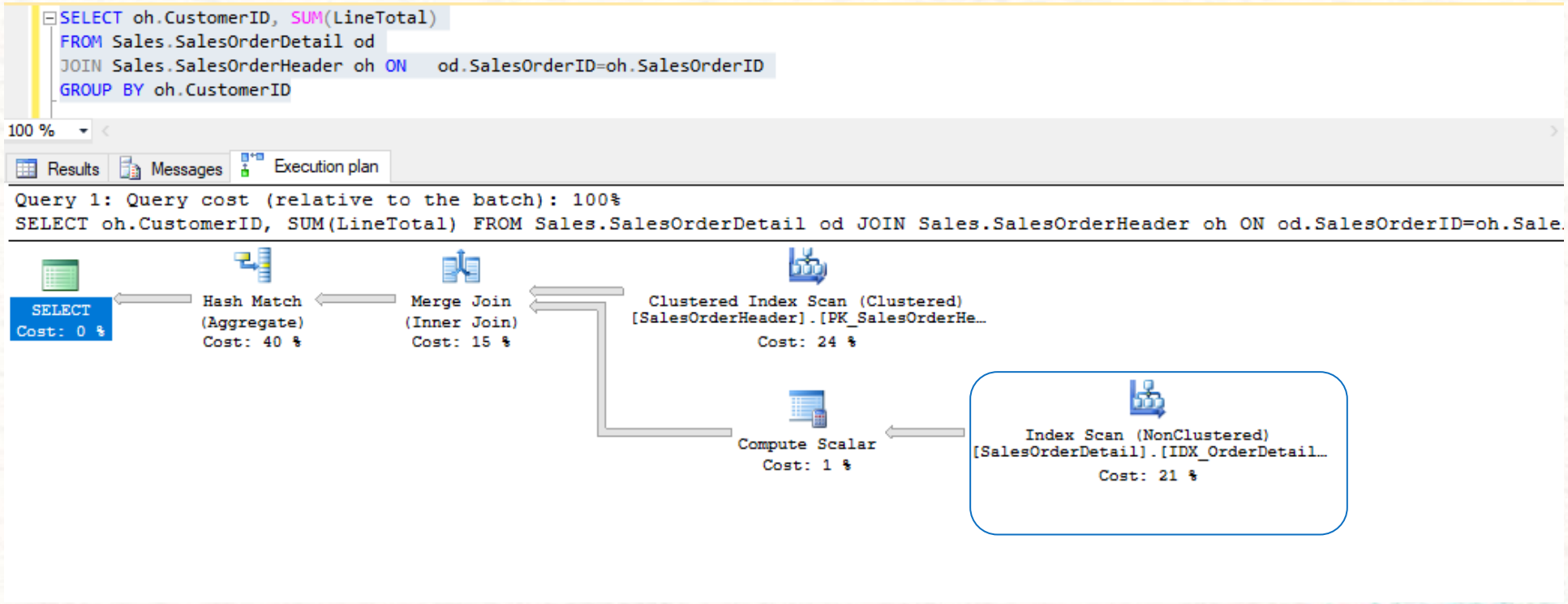


Plan de execuție grafic

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


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

Plan de execuție grafic

- Dacă interogarea este executată din nou după crearea indexului, putem vedea că indexul este folosit:



Proceduri stocate

Avantaje

- Avantaje de performanță
- Pe server
- Reutilizarea planului de execuție

Notă: cerințe pentru reutilizarea unui plan

- Reutilizarea planurilor nu este benefică întotdeauna

Proceduri stocate – sugestii pentru optimizare

SET NOCOUNT ON

- **Nu** se afișează numărul de înregistrări afectate
- **Reduce** traficul pe **rețea**

Utilizați numele schemei cu numele obiectului

- Ajută la găsirea **directă** a planului **compilat**

```
SELECT * FROM dbo.MyTable;
EXEC dbo.StoredProcedure;
```

Proceduri stocate – sugestii pentru optimizare

Nu utilizați prefixul sp_

- SQL Server caută întâi în baza de date *master* și ulterior în baza de date curentă

Evitați comparațiile de valori din coloane de tipuri diferite

- În cazul coloanei convertite, conversia implicită se va aplica asupra valorii din coloană pentru toate înregistrările din tabel (SQL Server trebuie să convertească toate valorile pentru a putea efectua comparația)

Proceduri stocate – sugestii pentru optimizare

Evitați join-urile între două tipuri de coloane

Utilizați UNION pentru a implementa o operație "OR“

Proceduri stocate – sugestii pentru optimizare

sp_executesql versus EXEC

- Planul de execuție al unei instrucțiuni **dinamice** poate fi **reutilizat** dacă **TOATE caracterele** pentru două execuții **consecutive** sunt **identice**

```
EXEC('SELECT * FROM Categories WHERE category_id=1;')

EXEC('SELECT * FROM Categories WHERE category_id=2;')


EXECUTE sp_executesql N'SELECT * FROM Categories WHERE

category_id=@category_id;', N'@category_id INT',

@category_id=1;
```

Proceduri stocate – sugestii pentru optimizare

Cursoare

În general **consumă mult din resursele SQL Server** și **reduc performanța și scalabilitatea aplicațiilor**

Sunt mai **indicate** când:

- Datele trebuie accesate înregistrare cu înregistrare / logică procedurală
- Se efectuează calcule **ordonate**

Proceduri stocate – sugestii pentru optimizare

Nu utilizați `COUNT()` într-o subinterogare pentru a verifica existența unor date

- Utilizați `IF EXISTS(SELECT 1 FROM table_name;)`

Output-ul instrucțiunii `SELECT` imbricate nu este folosit

- **Reduce** timpul de procesare și transferul pe rețea

Mențineți tranzacțiile scurte

- Lungimea tranzacțiilor influențează blocările și interblocările

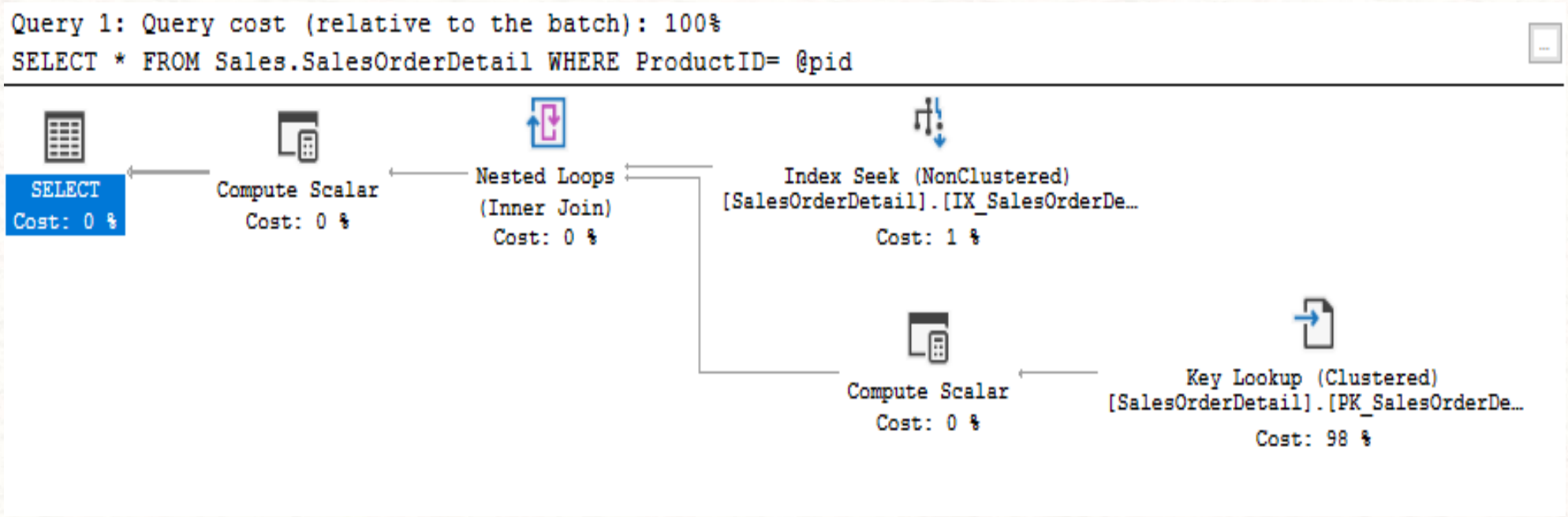
Proceduri stocate – sugestii pentru optimizare

- Reutilizarea planului de execuție

```
CREATE PROCEDURE usp_test (@pid INT)

AS

BEGIN

SELECT * FROM Sales.SalesOrderDetail WHERE
ProductID=@pid;

END;
```

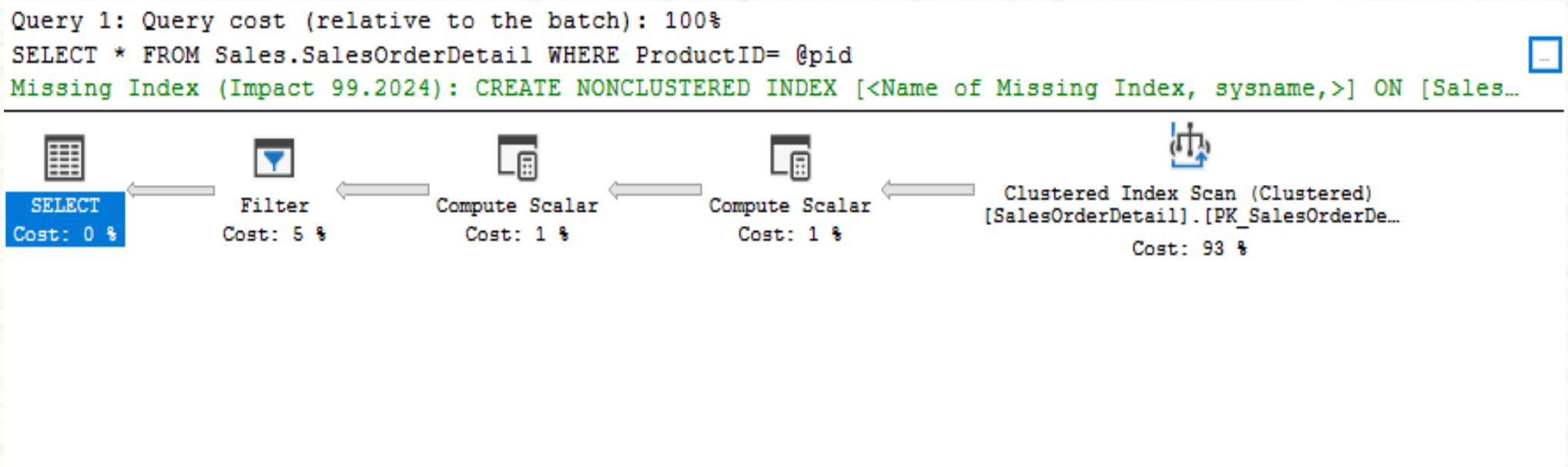
Proceduri stocate – sugestii pentru optimizare

```
EXEC usp_test 897;
```



Proceduri stocate – sugestii pentru optimizare

```
EXEC usp_test 870;
```



Proceduri stocate – sugestii pentru optimizare

- ***Hint***-urile în interogări specifică faptul că acestea ar trebui folosite pretutindeni în interogare și afectează **toți** operatorii din **instrucțiune**
- ***Hint***-urile sunt specificate în **clauza OPTION**
- Dacă optimizatorul (*query optimizer*-ul) generează un plan care **nu** este **valid** din cauza unor *query hints*, apare eroarea **8622**
- *Query hints* sunt recomandate spre a fi folosite **doar** ca **ultimă soluție** de către **programatori cu experiență** și **administratori de baze de date** (*SQL Server query optimizer* selectează de obicei cel mai **bun** plan de execuție pentru o interogare)

Proceduri stocate – sugestii pentru optimizare

- **OPTIMIZE FOR** *query hint* determină folosirea unei anumite valori pentru o variabilă locală la **compilarea** și **optimizarea** unei interogări
- Exemplu:

```
ALTER PROCEDURE usp_test (@pid INT)
AS
BEGIN
SELECT * FROM Sales.SalesOrderDetail WHERE ProductID=@pid
OPTION (OPTIMIZE FOR (@pid=870));
END;
```

Proceduri stocate – sugestii pentru optimizare

- **RECOMPILE** *query hint* determină **eliminarea** planului generat pentru o interogare **după** execuția acesteia, **forțând** optimizatorul să **recompileze** un plan de execuție data viitoare când **aceeași** interogare este **executată**
- Dacă **nu** se specifică **RECOMPILE**, *Database Engine* **salvează** în *cache* planurile de execuție și le **refolosește**
- La compilarea planurilor de execuție, **RECOMPILE** *query hint* **folosește** valorile **curente** ale variabilelor locale din interogare și, dacă interogarea se află în interiorul unei proceduri stocate, valorile **curente** ale **parametrilor**

Proceduri stocate – sugestii pentru optimizare

- **RECOMPILE query hint**
- Exemplu:

```
ALTER PROCEDURE usp_test (@pid INT)
AS
BEGIN
SELECT * FROM Sales.SalesOrderDetail
WHERE ProductID=@pid OPTION (RECOMPILE);
END;
```

Proceduri stocate – sugestii pentru optimizare

- **OPTIMIZE FOR UNKNOWN** determină folosirea de date statistice în locul valorilor inițiale pentru toate variabilele locale atunci când interogarea este compilată și optimizată, inclusiv parametrii creați cu parametrizare forțată
- Variabilele locale **nu** sunt cunoscute la optimizare
- Exemplul de mai jos generează întotdeauna același plan de execuție

```
ALTER PROCEDURE usp_test (@pid INT)
AS
BEGIN
SELECT * FROM Sales.SalesOrderDetail WHERE ProductID=@pid
OPTION (OPTIMIZE FOR UNKNOWN);
END;
```

Proceduri stocate – sugestii pentru optimizare

- Exemplul de mai jos generează întotdeauna același plan de execuție

```
ALTER PROCEDURE usp_test (@pid INT)
AS
BEGIN
DECLARE @lpid INT;
SET @lpid=@pid;
SELECT * FROM Sales.SalesOrderDetail WHERE ProductID=@lpid;
END;
```

Proceduri stocate – sugestii pentru optimizare

- Exemplu:

```
DECLARE @city_name VARCHAR(30);
DECLARE @postal_code VARCHAR(15);
SELECT * FROM Person.Address
WHERE City=@city_name AND PostalCode=@postal_code
OPTION (OPTIMIZE FOR (@city_name='Seattle',
@postal_code UNKNOWN));
```

Proceduri stocate – sugestii pentru optimizare

Alte *hint*-uri pentru interogări (*query hints*)

- **HASH GROUP** vs **ORDER GROUP**
- Exemplu:

```
SELECT ProductID, OrderQty, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
GROUP BY ProductID, OrderQty
ORDER BY ProductID, OrderQty
OPTION (HASH GROUP, FAST 10);
```

Proceduri stocate – sugestii pentru optimizare

Alte hint-uri pentru interogări

- **MERGE UNION** vs **HASH UNION** vs **CONCAT UNION**
- Exemplu:

```
SELECT * FROM HumanResources.Employee AS E1
UNION
SELECT * FROM HumanResources.Employee AS E2
OPTION (MERGE UNION);
```

Proceduri stocate – sugestii pentru optimizare

Hint-uri pentru **join**

- **LOOP JOIN** vs **MERGE JOIN** vs **HASH JOIN**
- Exemplu:

```
SELECT * FROM Sales.Customer AS C
INNER JOIN Sales.vStoreWithAddresses AS SA
ON C.CustomerID=SA.BusinessEntityID
WHERE TerritoryID=5
OPTION (MERGE JOIN);
GO
```

Proceduri stocate – sugestii pentru optimizare

Hint-uri pentru **join**

- FAST n determină returnarea primelor **n** înregistrări cât de **repede** este posibil
- După ce sunt returnate primele **n** înregistrări, interogarea își *continuă* execuția și **produce** întregul *result set*
- Exemplu:

```
SELECT * FROM Sales.Customer AS C
INNER JOIN Sales.vStoreWithAddresses AS SA
ON C.CustomerID=SA.BusinessEntityID
WHERE TerritoryID=5
OPTION (FAST 10);
```

Proceduri stocate – sugestii pentru optimizare

Hint-uri pentru **join**

- FORCE ORDER –"forțează" optimizatorul să utilizeze ordinea *join*-urilor ca în interogare

```
SELECT * FROM Table1
INNER JOIN Table2 ON Table1.a=Table2.b
INNER JOIN Table3 ON Table2.c=Table3.d
INNER JOIN Table4 ON Table3.e=Table4.f
OPTION (FORCE ORDER);
```

Proceduri stocate – sugestii pentru optimizare

- Mai multe despre **controlarea** planurilor de execuție cu ajutorul *hint*-urilor:
- https://www.simple-talk.com/sql/performance/controlling-execution-plans-with-hints/

Tabele temporare

Utile când:

- Aveți *result set*-uri **intermediare** care trebuie să fie accesate de mai **multe** ori
- Aveți nevoie de o **zonă de stocare temporară** pentru date în timp ce executați cod **procedural**

Utilizați tabelele temporare când:

- Lucrați cu **volume mari de date**, iar **eficiența** planurilor este **importantă** și **netrivială**

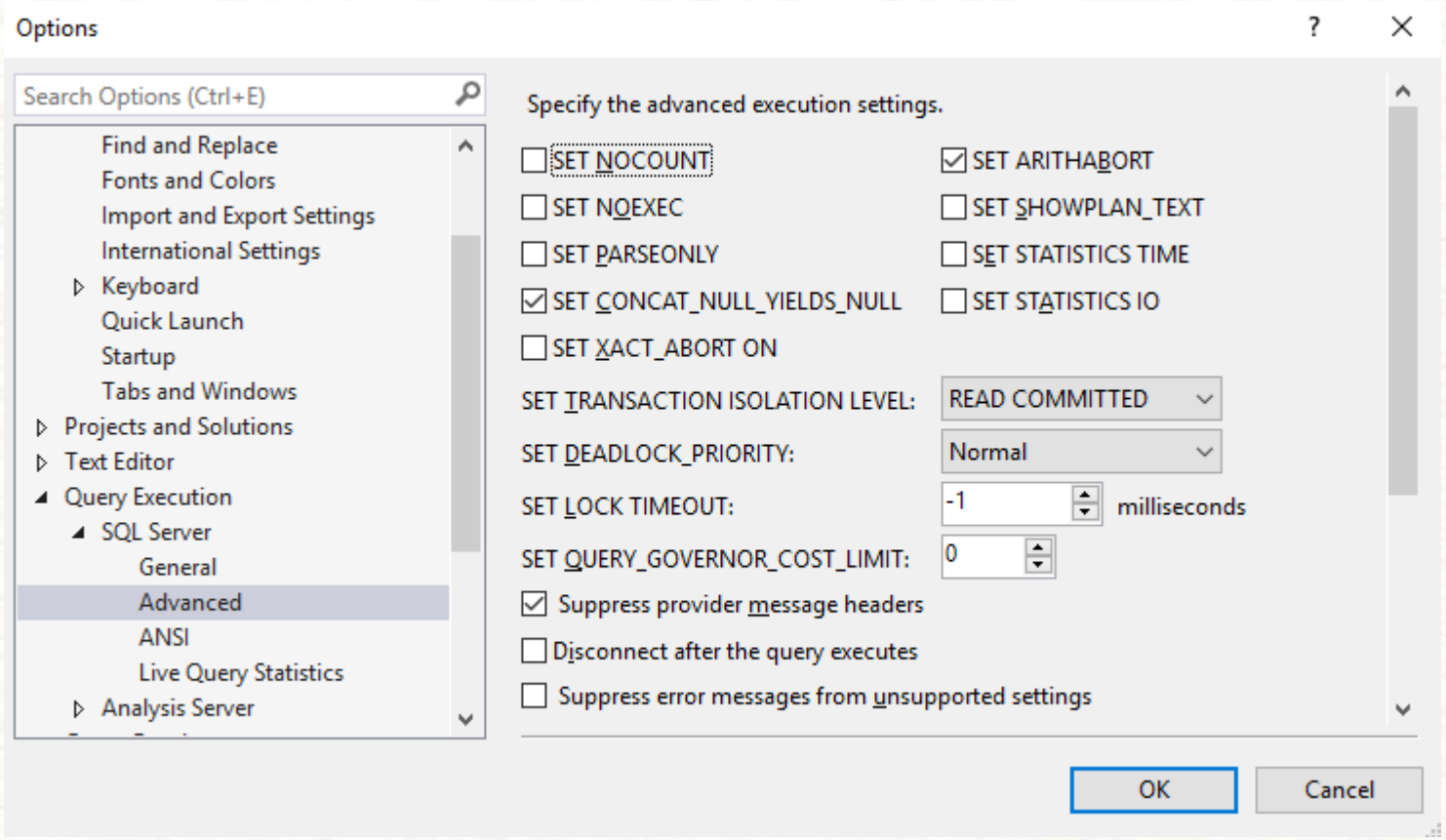
Utilizați variabile tabel când:

- Lucrați cu **volume mici de date**, iar **eficiența** planurilor **nu** este atât de importantă ca **recompilările**, sau atunci când planurile sunt **triviale**

Triggers

- **Costisitoare** (when rollback, undo as opposed to reject)
- Impactul **mare** asupra **performanței** implică accesarea tabelelor speciale *inserted* și *deleted*:
  - SQL Server 2000: transaction log
  - SQL Server 2005: row versioning (*tempdb*)
- Încercați să utilizați *set-based* activities
- Identificați numărul de înregistrări afectate și reacționați în consecință
- UPDATE triggers înregistrează delete urmat de insert în *log*

Opțiuni SQL Server



Fragmentare

Fragmentarea are un efect **important** asupra **performanței** interogărilor

- Fragmentare **logică**: procentul de pagini *out-of-order*
- **Densitatea** paginilor: popularea paginilor

Utilizați **DBCC SHOWCONTIG** pentru a obține statistici legate de fragmentare și examinați **LogicalFragmentation** și **Avg. Page Density (full)**

Utilizați funcția **sys.dm_db_index_physical_stats** și analizați **AvgFragmentation**

Reconstruiți indecșii pentru a gestiona fragmentarea

Alte statistici

Update statistics asynchronously

- **String summary statistics**: frecvența distribuției subșirurilor este menținută pentru coloanele care stochează șiruri de caractere
- **Asynchronous auto update statistics** (setat implicit pe off)
- Statistici pentru coloane calculate

Dynamic management view-ul sistem *sys.dm_exec_query_stats*

- Returnează statistici legate de performanță pentru planurile de execuție din *cache*
- Conține câte o înregistrare pentru fiecare *query statement* din planul aflat în *cache*, iar durata de existență a înregistrărilor este legată de cea a planului
- Când un plan este șters din *cache*, înregistrările care îi corespund sunt eliminate

Alte statistici

- **total_logical_reads / total_logical_writes** – numărul total de citiri / scrieri logice efectuate la execuțiile unui plan de când a fost compilat
- **total_physical_reads** – numărul total de citiri fizice efectuate la execuțiile unui plan de când a fost compilat
- **total_worker_time** – timpul CPU total utilizat, în microsecunde, pentru execuțiile unui plan de când a fost compilat
- **total_elapsed_time** – durata totală, în microsecunde, pentru execuțiile încheiate ale unui plan