Mai multe despre optimizare

Seminar 6

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)

Evitați join-urile între două tipuri de coloane

- Indecșii nu sunt utilizați pentru coloanele convertite

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

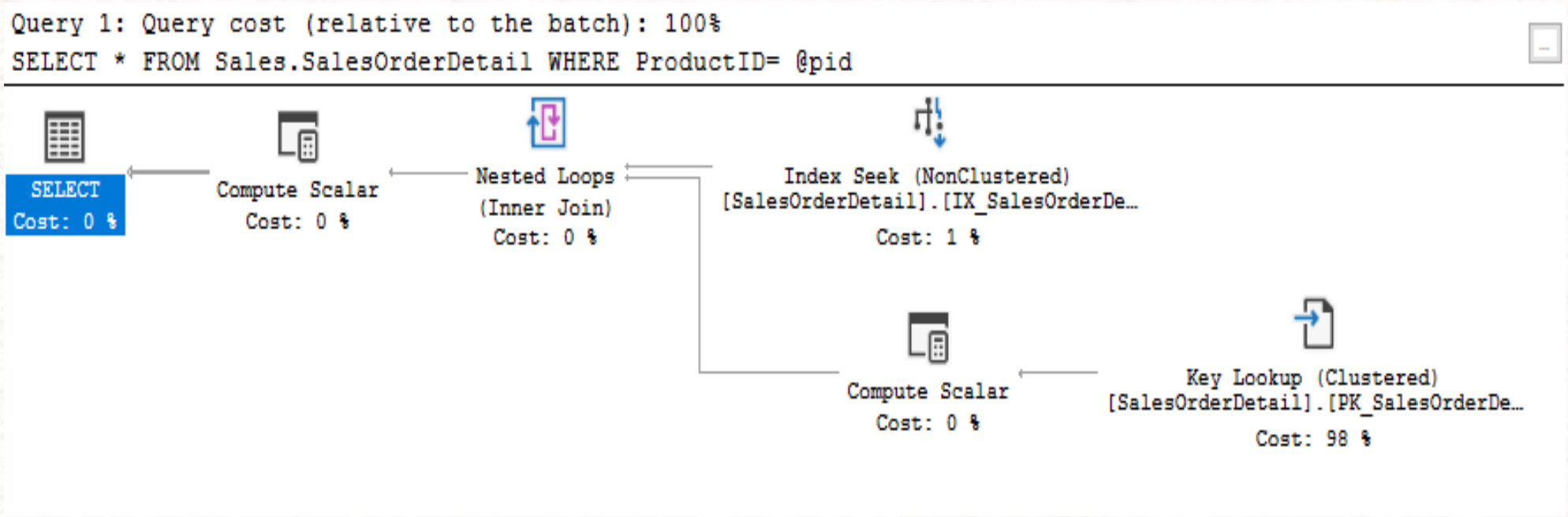
Proceduri stocate – sugestii pentru optimizare

- Reutilizarea planului de execuție

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

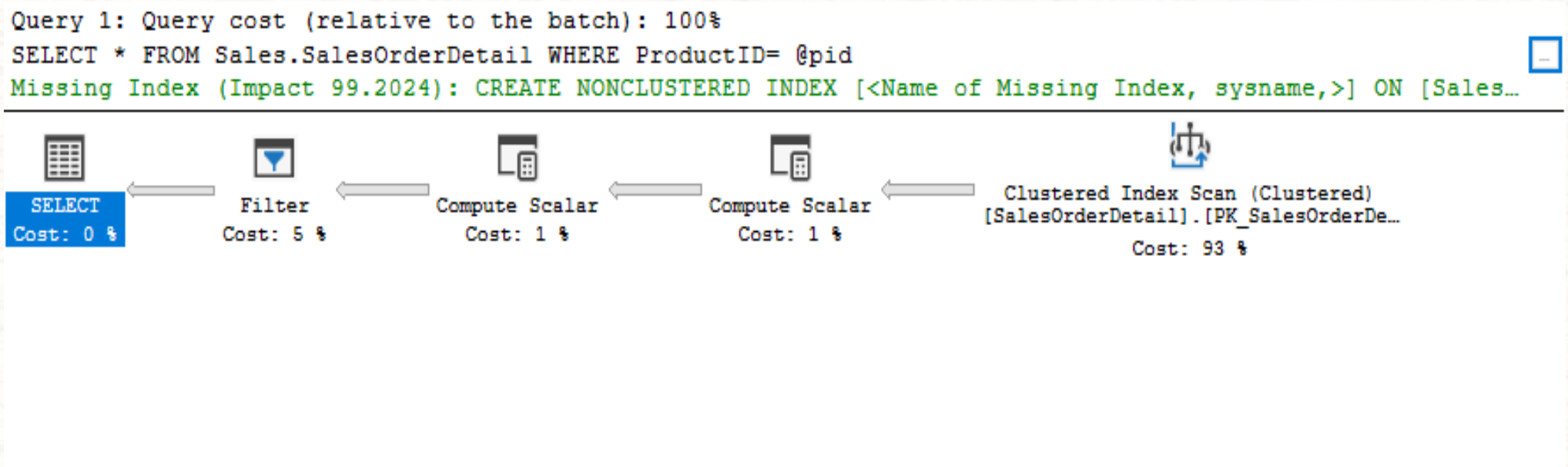
Proceduri stocate – sugestii pentru optimizare

```
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/

Execuția dinamică

Dezavantaje

- Cod urât care este dificil de întreținut
- Risc de securitate - SQL Injection

Utilizare inteligentă

- Sortare și filtre dinamice pentru a obține planuri bune
- Și altele…

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

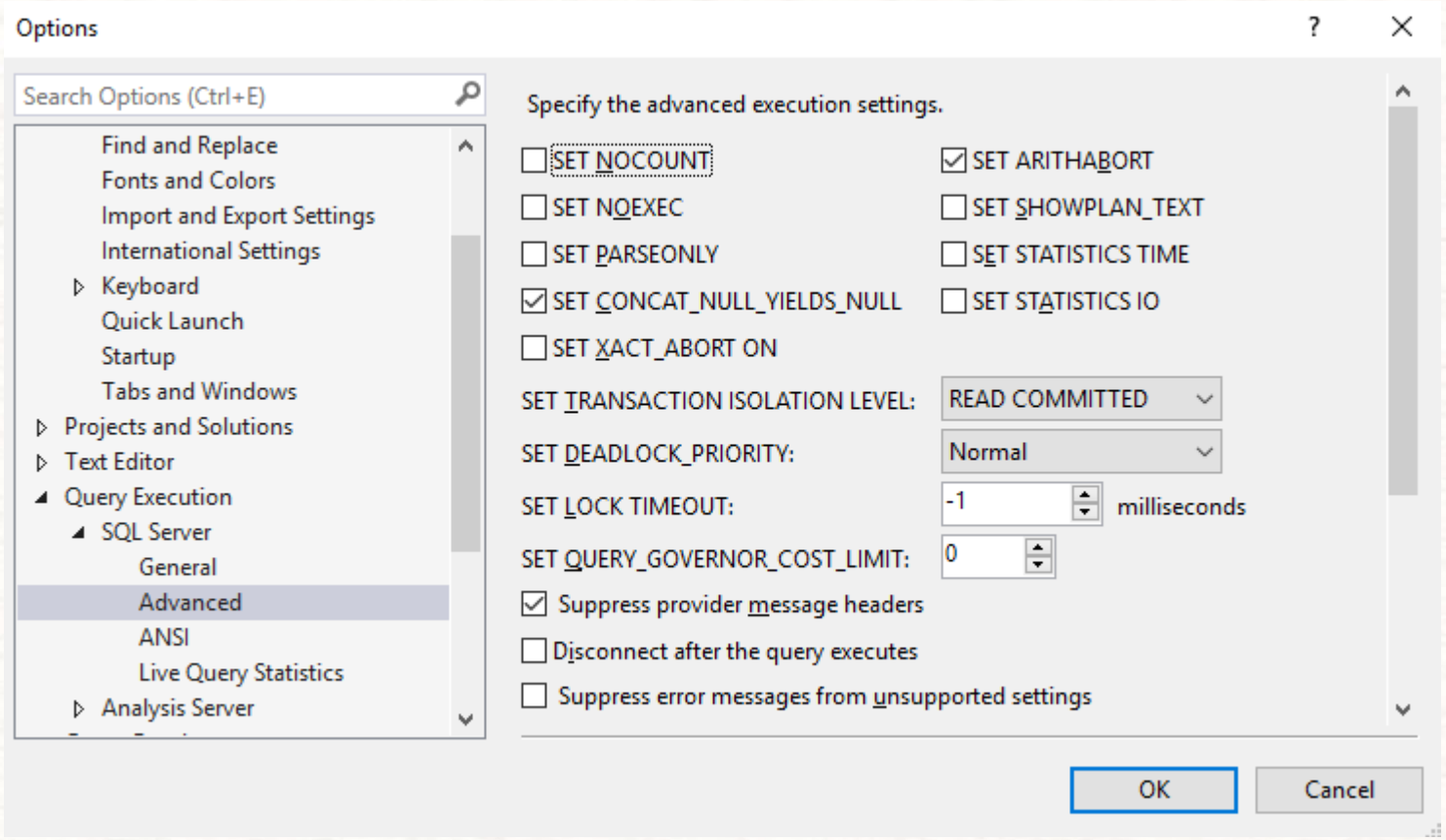
Utilizați variabile tabel când:

- 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