MsSQL: Basal performance tuning, part 3

Kort om stored procedures og rekompileringer.

Skrevet den **03. Feb 2009** af **trer** I kategorien **Databaser / MS SQL** | ⭐⭐⭐☆☆

Historik:
11 jan 2005: Info om tabelvariabler opdateret.

Disse artikler om basal performance tuning er uddrag af en intern anbefaling om SQL som jeg har skrevet til udviklerne på min arbejdsplads, NNIT a/s.

Fordelen ved stored procedures og functions frem for views og ad hoc SQL er, at man normalt kan spare kompilering og valg af en query plan for sin SQL.

Da disse to trin er en væsentlig del af CPU belastningen gælder det om at undgå situationer hvor SQL Server er tvunget til at rekompilere en given stored procedure.

Rekompileringer giver kort sagt dårlig performance og forhøjet serverbelastning idet rekompileringerne vil ske for hver gang hver enkelt bruger kalder en given stored procedure.

Kald aldrig egne stored procedures for SP_xxxx
Når en stored procedures navn starter med SP_ vil optimizeren først forsøge at finde den kaldte procedure i MASTER databasen. Findes den ikke der søges der så i aktuelle database.

Ved at navngive stored procedures uden SP_ prefixet vil søgningen kunne begrænses til aktuelle database og rekompilering undgås.

Prefix altid stored procedures med ejer
Ejer-prefixet sikrer hurtigere tilgang til det søgte objekt og tillader desuden optimizeren at genbruge eksekveringsplaner således at en ny plan ikke skal genereres og dermed at proceduren skal rekompileres for hver bruger der afvikler denne.

Undgå ANSI settings i stored procedures
ANSI indstillinger består af flagene ANSI_DEFAULTS, ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS og CONCAT_NULL_YIELDS_NULL.

Såfremt en eller flere af disse indstillinger ændres i en stored procedure - og ændringen gør, at der er en forskel til klientens ANSI indstillinger, så vil den pågældende stored procedure skulle rekompileres for hvert kald fra hver klient.

Undlad derfor at ændre ANSI settings fra database default.

Skal ANSI settings ændres bør det være på database niveau, og klientapplikationen (eller webserveren) skal sættes til samme ANSI settings for at undgå kontinuerlig rekompilering.

Overhold altid datatyper mellem parameter & kolonner
Sikr altid at datatyperne er ens mellem parametre og kolonner.

I SQL Server version 7 kan optimizeren ikke anvende indeks såfremt man bruger en varchar parameter

mod en nvarchar kolonne - og vice versa.

Undgå dynamisk SQL i stored procedures
Stored procedures der indeholder dynamisk sql - specielt i forbindelse med DDL (Data Definition Language - fx CREATE) - vil altid skulle rekompileres.

Dynamisk SQL bør forbeholdes specielle procedurer der vil blive kaldt af få brugere.

DDL og DML udtryk i stored procedure
Når DDL (Data Definition Language - fx CREATE eller DROP) og DML (Data Manipulation Language - fx INSERT eller DELETE) udtryk indgår i en stored procedure vil der altid ske rekompilering.

Undgå derfor at blande DDL (Data Definitions Language) og DML (Data Manipulation Language) udtryk, og placer altid samtlige DDL udtryk i starten af en stored procedure.

Korrekt SQL:

```
CREATE TABLE #TMP_TBL1 (f1 int, f2 int)
CREATE TABLE #TMP_TBL2 (f1 int, f2 int)
INSERT INTO #TMP_TBL1 (f1,f2) VALUES (1,1)
INSERT INTO #TMP_TBL2 (f1,f2) VALUES (1,1)
```

Fejlagtig SQL

```
CREATE TABLE #TMP_TBL1 (f1 int, f2 int)
INSERT INTO #TMP_TBL1 (f1,f2) VALUES (1,1)
CREATE TABLE #TMP_TBL2 (f1 int, f2 int)
INSERT INTO #TMP_TBL2 (f1,f2) VALUES (1,1)
```

I ovenstående eksempel vil den kaldende stored procedure blive rekompileret 1 gang i det korrekte udtryk og 2 gange i det fejlagtige udtryk.

Overvej om oprettelsen og arbejdet på de temporære tabeller kan undlades ved i stedet at anvende permanente tabeller - eller om man kan anvende tabel-variabler (SQL Server 2000).

Tabel-variabler erklæres og bruges således:

```
declare @mytable (
    kolonne int,
    .....
    kolonne varchar(50)
    )
insert into @mytable (kolonne,,,kolonne)
values (1,,,'tekst')
select kolonne,,,kolonne
from @mytable
```

Fordelen ved tabelvariabler er, at de giver færre rekompileringer og kræver færre låse- og logressourcer end temporære tabeller. Der er dog grænser for hvilke operationer man kan foretage på tabel-variabler.

Bemærk dog at alle query planer på tabelvariabler kun forventer 1 række i en tabelvariabel. Det betyder at den med "mange" rækker vil kunne vælge en suboptimal tilgang til data ved joins. Eneste mulighed man har for at afhjælpe dette problem er ved join-hints.

Data i stored procedures temporære tabeller
Vær opmærksom på de forventede datamængder i temporære tabeller der benyttes i stored procedures.

SQL Server forventer små datamængder i temporære tabeller og er meget aggressiv mht statistik opdateringer ved ændringer. Statistikkerne opdateres for hver 6 rækker ændret i den temporære tabel.

Hver statistik opdatering vil fremtvinge en rekompilering af alle stored procedures der benytter den temporære tabel.

Tilgå derfor altid temporære tabeller med option hint KEEP PLAN sat. Dermed vil SQL Server benytte samme frekvens for statistik opdateringer som benyttes for almindelige tabeller, dvs. ved 20% ændringer i tabellen. Se i øvrigt Microsoft Books Online vedr. brug af OPTION (KEEP PLAN)

Korrekt SQL:

```
SELECT f1, f2 FROM #TMP_TBL1 OPTION (KEEP PLAN)
UPDATE #TMP_TBL1 SET f1=1 WHERE f2=2 OPTION (KEEP PLAN)
DELETE FROM #TMP_TBL1 WHERE f2=2 OPTION (KEEP PLAN)
```

Overvej altid om en permanent tabel bør anvendes frem for en temporær for at sænke antal rekompileringer.

Begræns kommunikation mellem server og klient
Indsæt altid udtrykket SET NOCOUNT ON i starten af en stored procedure eller trigger.

Udtrykket undertrykker SQL Servers normale tilbagemelding om antal rækker berørt / opdateret etc. hvormed der spares et roundtrip fra klient til server.

God fornøjelse
Troels

Kommentar af driis d. 19. Dec 2004 | 1

Udmærket, jeg har lært noget.

Kommentar af arne_v d. 18. Dec 2004 | 2

Udmærket.