Kapitel 11 – SQL Optimering

Hardware optimering

CPU overvejelser

For de fleste databaser er én enkelt dual eller quadcore processor ganske udemærket, men det giver god mening at have et motherboard med et ekstra socket i tilfælde af at dine aktive forbindelser og mængden af queries stiger til et niveau hvor 2 processorer er nødvendigt.

L2 Cache er vigtigt, at gå fra f.eks 512KB til 2MB kan give en forskel i performance på omkring 20%

Hvis du har en processor med HT (2 virtuelle processorer i én) er det en god idé at disable det da den virtuelle context switching sløver cpu'en.

Hukommelse

Hukommelse er det vigtigste at overveje.
Nu mere jo bedre!
Eskalér mængden af RAM efter behov.
64bit server supporterer 32GB til 2TB RAM
32 bit er begrænset til 4gb, nomindre du har en DataCenter Edition (128GB) eller en Enterprise (64GB)

Storage

At adskille dine Logfiles og Databaser og selve installationene ud på forskellige Raid's kan give et performance boost på op til 30%.
At ligge sin tempdb på sit eget drev kan også booste din performance.
Dette skyldes at mængeden af IOPS (I/O Operations per second) per drive er begrænset, så at ligge dine data ud på forskellige lokationer gør at denne flaskehals minimeres.
Writecaching kan give ekstra performance men skal deaktiveres da de data den gemmer midlertidigt før den skriver dem kan mistes ved power failure nomindre de supporterer battery check for deres write chache.
Solid State hvis du har råd :D

Netværks Design

I de fleste tilfælde vil et enkelt 100Mbs / 1Gbs kort du fint.
Set advarsler på i det tilfælde at den ikke kan følge med.

Virtualisering

Er din database meget I/O intens er det en dårlig idé at virtualisere den da du kan miste over 20% performance.
For mindre database operationer er dette dog acceptabelt hvis de andre overvejelser nævnt her på selve jernet er i orden.
En Virtualiseret SQL server er ret let at flytte så i tilfælde af at din performance dropper for meget kan det måske betale sig at flytte den over på en host der ikke har helt så travlt.

Design Overvejelser

Ydedygtige tabeller

Normalization af dine tabeller er vigtigt.

Har du mange meget data i din tabel som går igen og igen er det en god idé at separerer dem ud i multiple tabeller og binde databaserne sammen med Primary og Foreign keys

ALTER TABLE %schema%.%databasenavn% ADD PRIMARY KEY (%tabelnavn%)

ALTER TABLE %schema%.%databasenavn% ADD FOREIGN KEY (%tabelnavn%) REFERENCES %eksternschema%.%eskterndatabase%(%eksterntabel%)

NavnID	Navn	TeleArealD	TeleID
1	Brian	1	1
2	Curth	2	2
3	Voleck	1	3

TeleArealD	AreaNumber
1	+457585
2	+456550

TeleID	TeleNumber
1	5690
2	5421
3	6789

De gule felter er Primary keys og de blå er foreign.

I dette tilfælde har Brian f.eks telefonnummer +4575855690, da han bor i samme by som Voleck kan vi genbruge denne data. Voleck har nummer +4575856789

Datatyper

Char versus varchar – varchar kan eskalerer sig hvorimod char optager en predifineret størrelse hele tiden.

Brug unicode strings hvis du vil have internationale karakterer med i din database.

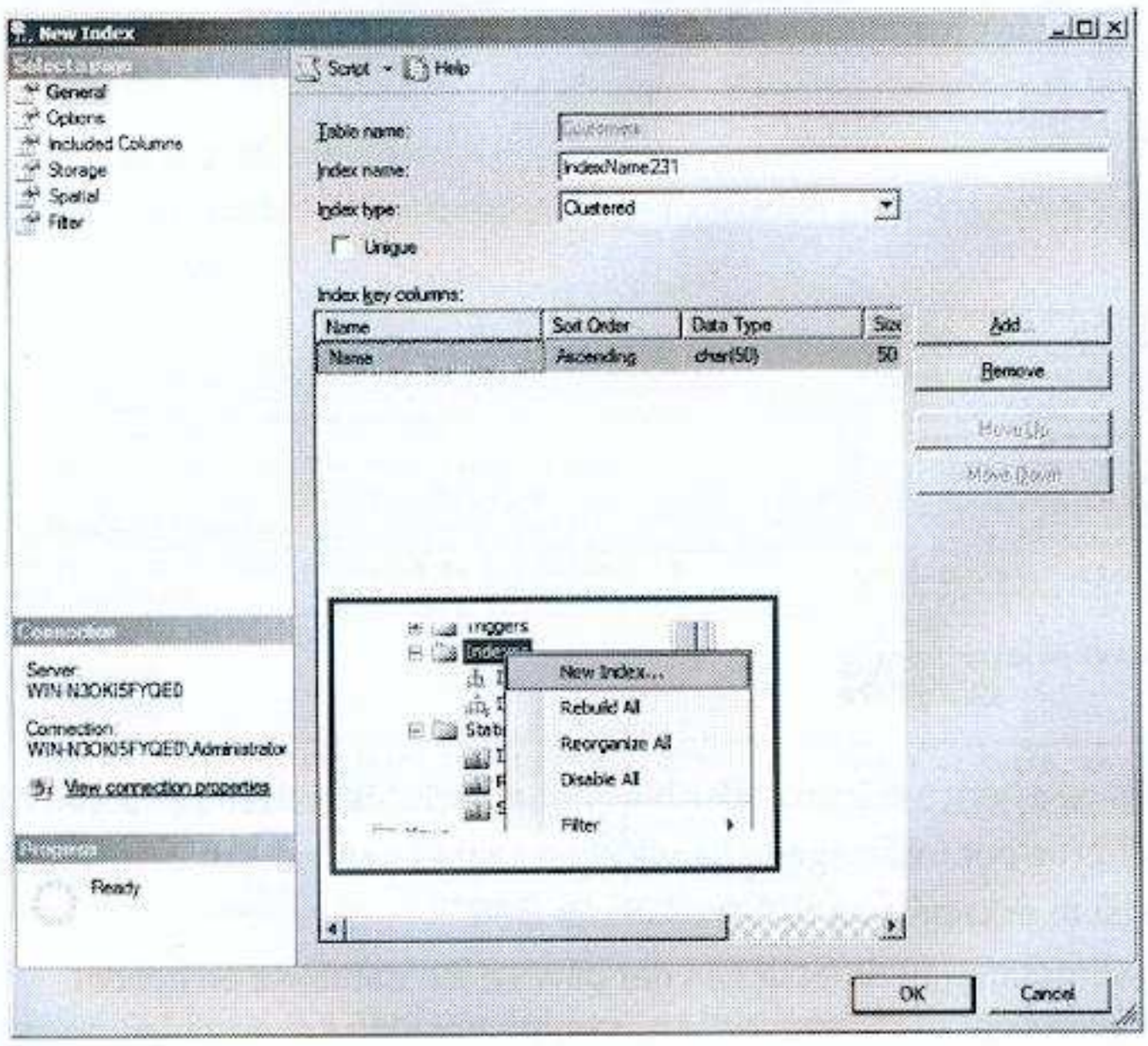
Integers: tinyint(byte)= 0 – 256 range, smallint(Int16) = -32768 – 32767 range – brug den mindste integer du kan!

Generelt minimér den mængde af data du holder i hver row – husk at hvis du har defineret en stor datamængde men kun bruger en lille del flyder denne lille del rigmelig meget ;p

Indexing - og af hvad?

Indexing er smart, men minimér din indexing til de felter du véd der vil komme mange ordered forspørgelser på.

I overstående eksempel ville det måske kun være "Navn" og "AreaNumber".



Minimering og overvågning af Blocks

Blocks er nasty!

Blocks betyder at hver gang en applikation skal have fat i databasen bliver den row den vil tilgå blocket i det splitsekund den tilgår det i.

Der findes en række af blocks alt efter hvad applikationen vil med row'en.

Under normale omstændigheder tales der om 2 block modes.

Shared (Read block)

Exclusive (Write Blocks)

Hvis en applikation vil læse i en row får den en shared block, det vil sige at andre applikationer stadig kan læse i den men ikke skrive til den i den tid applikationen har fat i dataen.

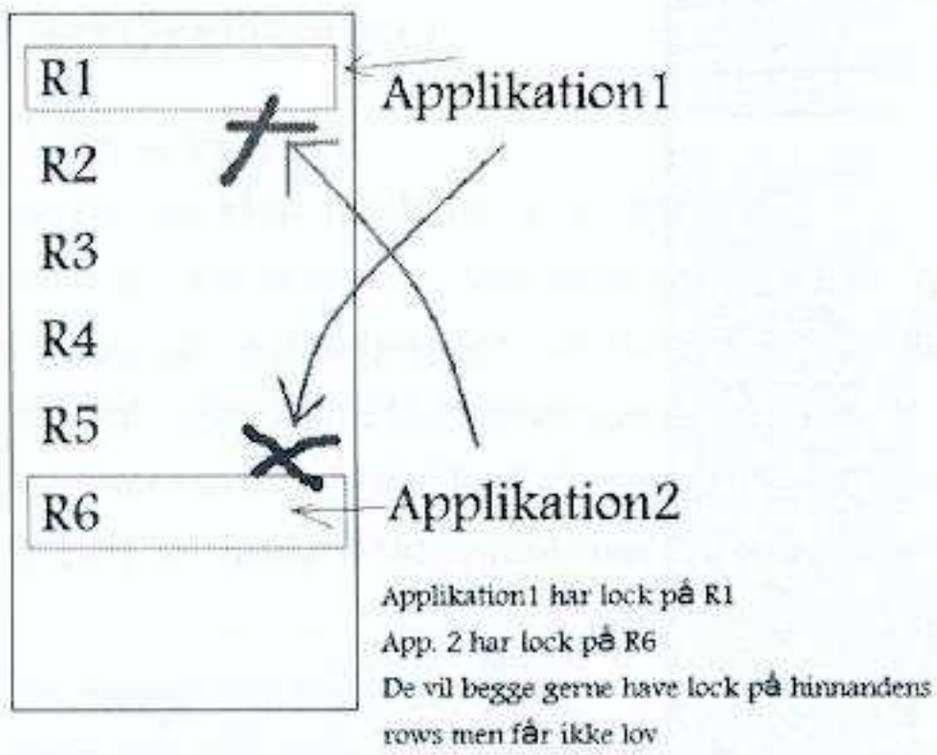
Hvis den samme applikation så vil skrive til den får den en Exclusive block som gør at ingen kan tilgå dataen før apllikationen er færdig med at skrive dataen.

Data i SQL er organiseret i Pages inde i tabellen. Hvis en applikation har writelock i over halvdelen af page'en giver SQL server applikationen en Page-Lock, denne lock eskaleres efterhånden som applikationen får fat i mere og mere (sekventiel) data – hele vejen op til en fuld database-lock goså selvom applikationen måske ikke bruger alt data i tabellen.

Dette er måske fint hvis det kun er i et lille split sekund, men hvis du ser længere og længere lock-waits og måske endda time-outs skal du måske se på hvad det er applikationen laver og hvorfor den ikke slipper sien locks.

En anden situation er Deadlocks (se nedenstående tegning).

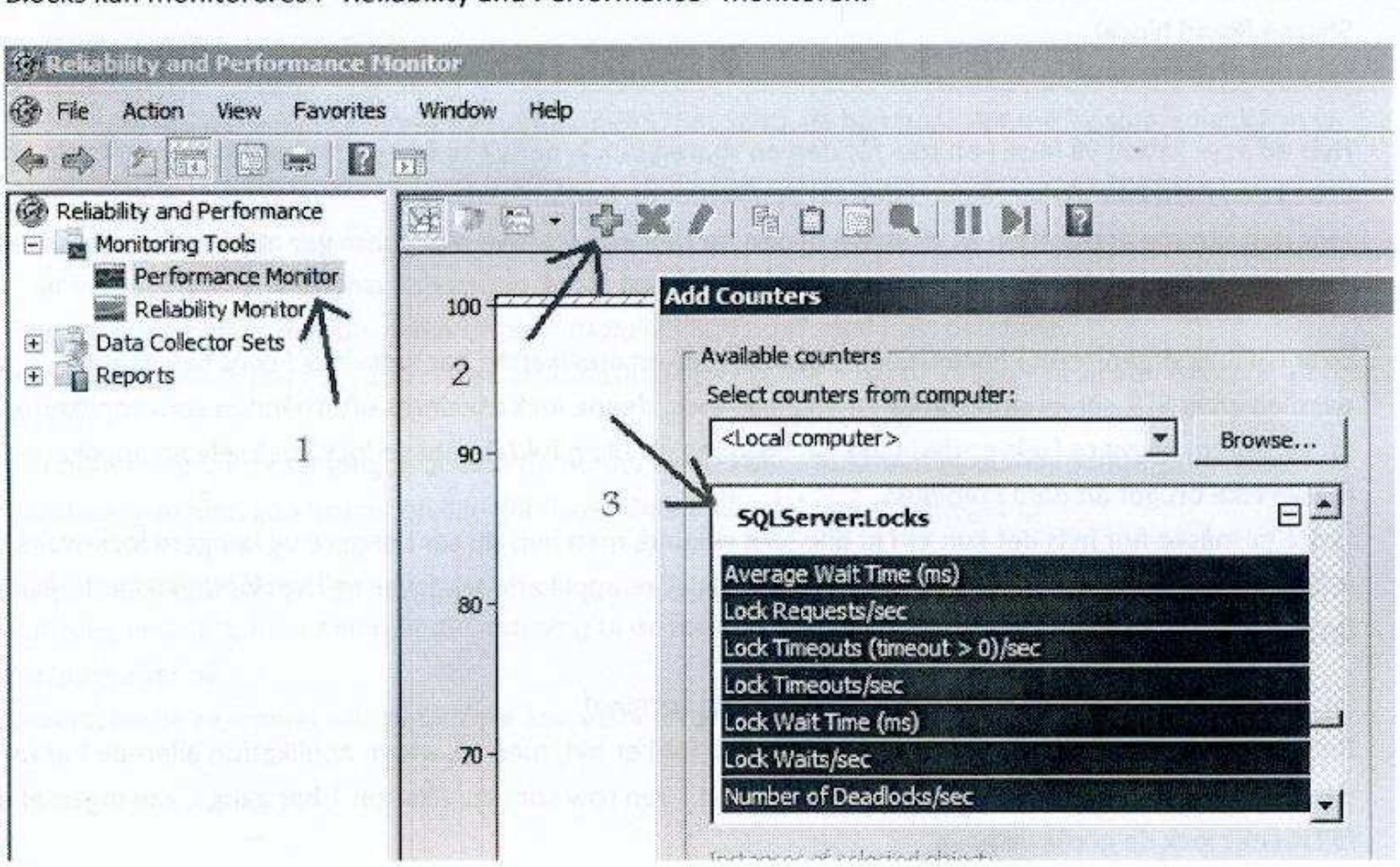
Hvis en application har fat i et felt og vil til at have fat i et nyt, men en anden applikation allerede har fat i denne og applikation nummer to gerne vil have fat i den row som applikation 1 har gang i, kan ingen af dem få fat i det lock de gerne vil have.



SQL server scanner selv efter deadlocks og efter ca 0.5 sekunder bliver et af locks'ne slettet og dataen fries op, Deadlocks max tiden for applikationerne der forårsagede deadlocken sættes ned indtil at den stopper med at deadlocke og den bliver så langsom eskaleret op til 0.5 sekunder igen.
Men igen, ser du mange deadlocks og ventetider og timeout kan det påvirke din database en hel del.

Normaliserede tabeller, stored procedues og korte transaktioner kan nedsætte mængden og den tid der bruges på blocks.
Sørg for at du evaluerer, slår altnødvendig data op og først derefter skriver alle ændringerne på én gang.
På denne måde vil du få flere read locks og færre write blocks.

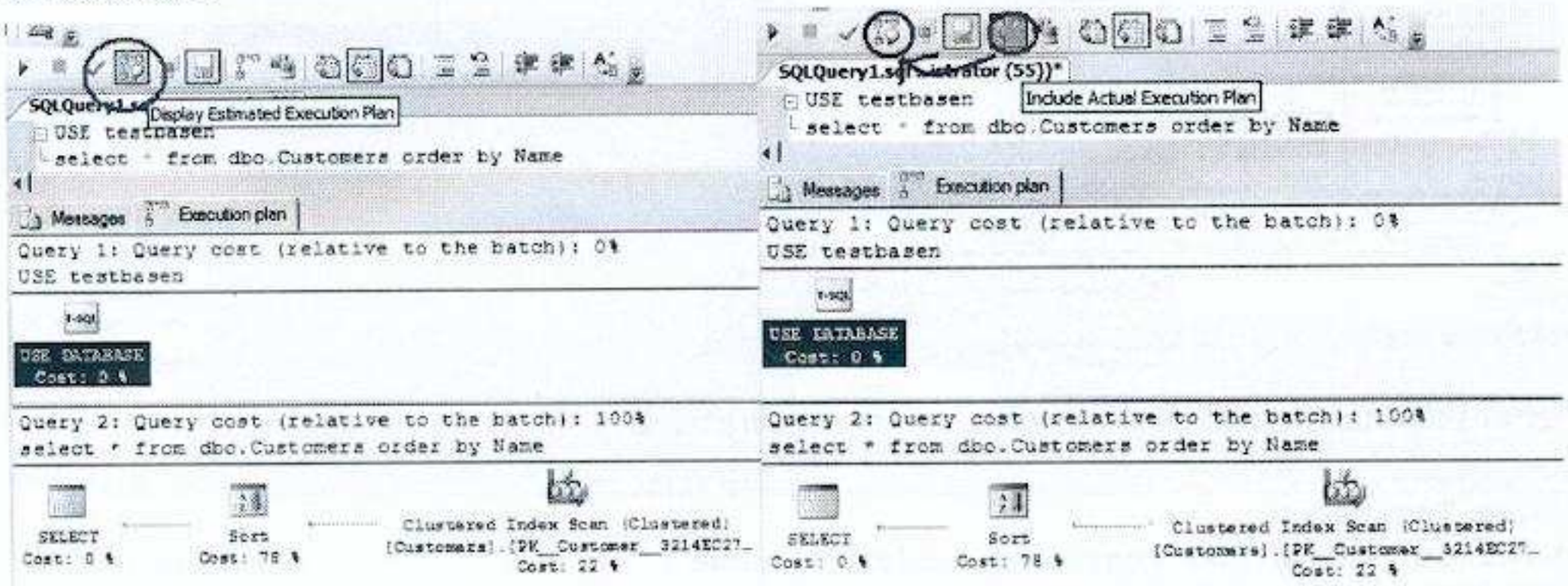
Blocks kan monitoreres i "Reliability and Performance" monitoren.



Query Optimering

Opdatering af Statistics

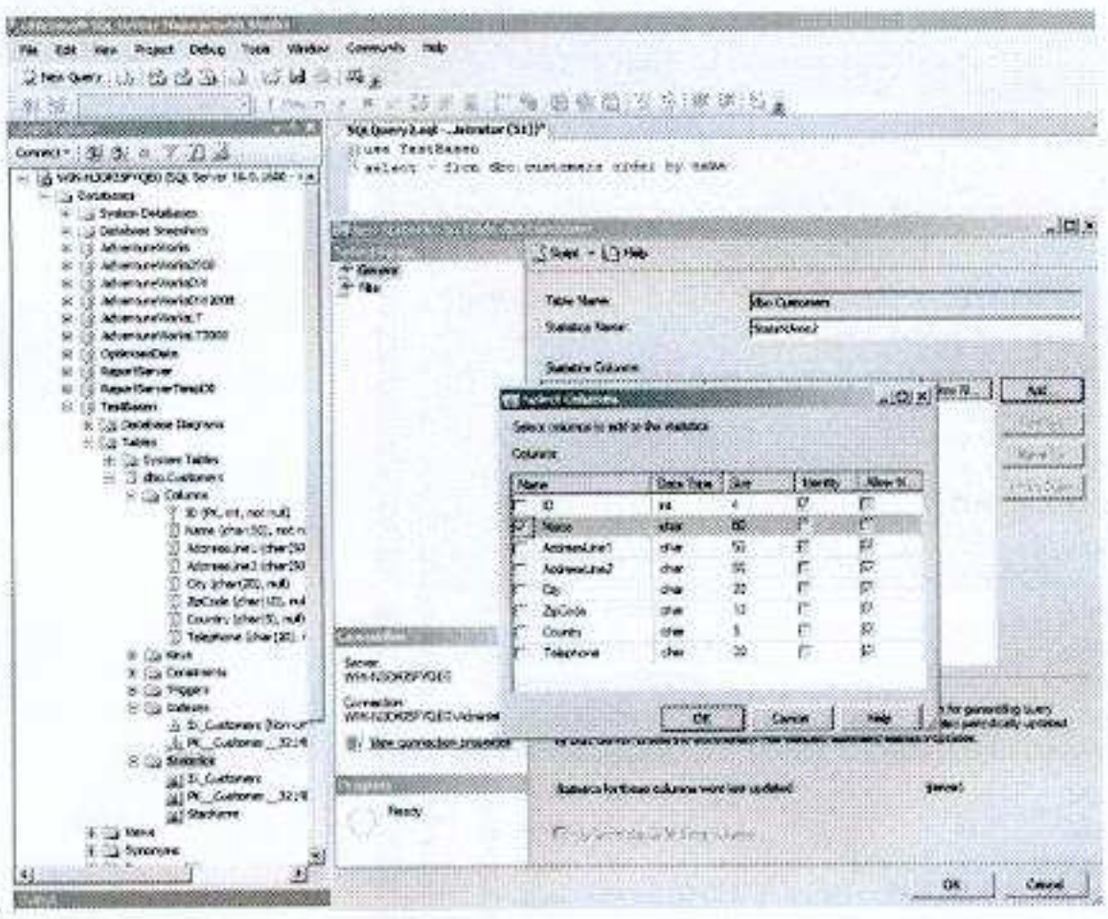
Hvis der er forskel på dine estimerede execution planer og dine reelle execution skal du nok have opdateret dine statistcs.



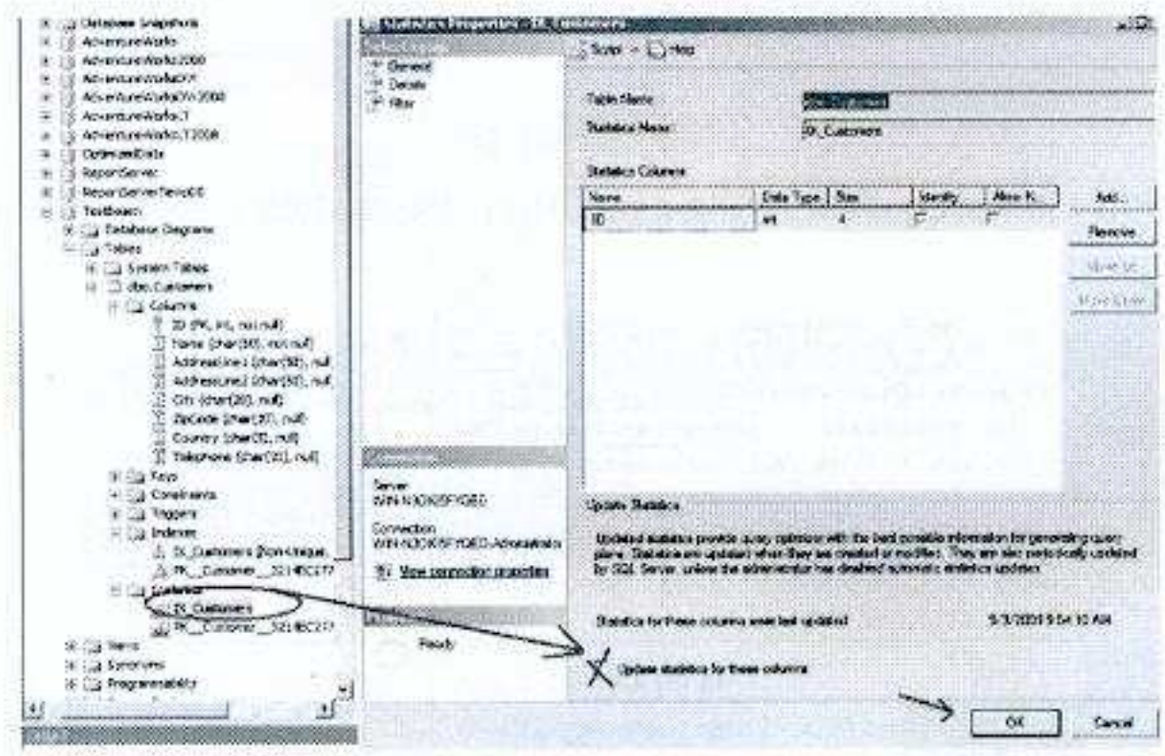
Statistics holder øje med om det f.eks er hurtigere at bruge indexet end at selv sorte databasen.

Sørg for at have et index af det du vil sorte.

Opret en statistic



Derefter kan du opdatere den efter behov som set på næste billede.



Derefter skulle dine executionplans gerne se meget bedre ud :D

```
use TestBasen
select * from dbo.customers order by name
```

Messages | Execution plan | Client Statistics

```
Query 1: Query cost (relative to the batch): 0%
use TestBasen
```

```
USE DATABASE
  Cost: 0 %
```

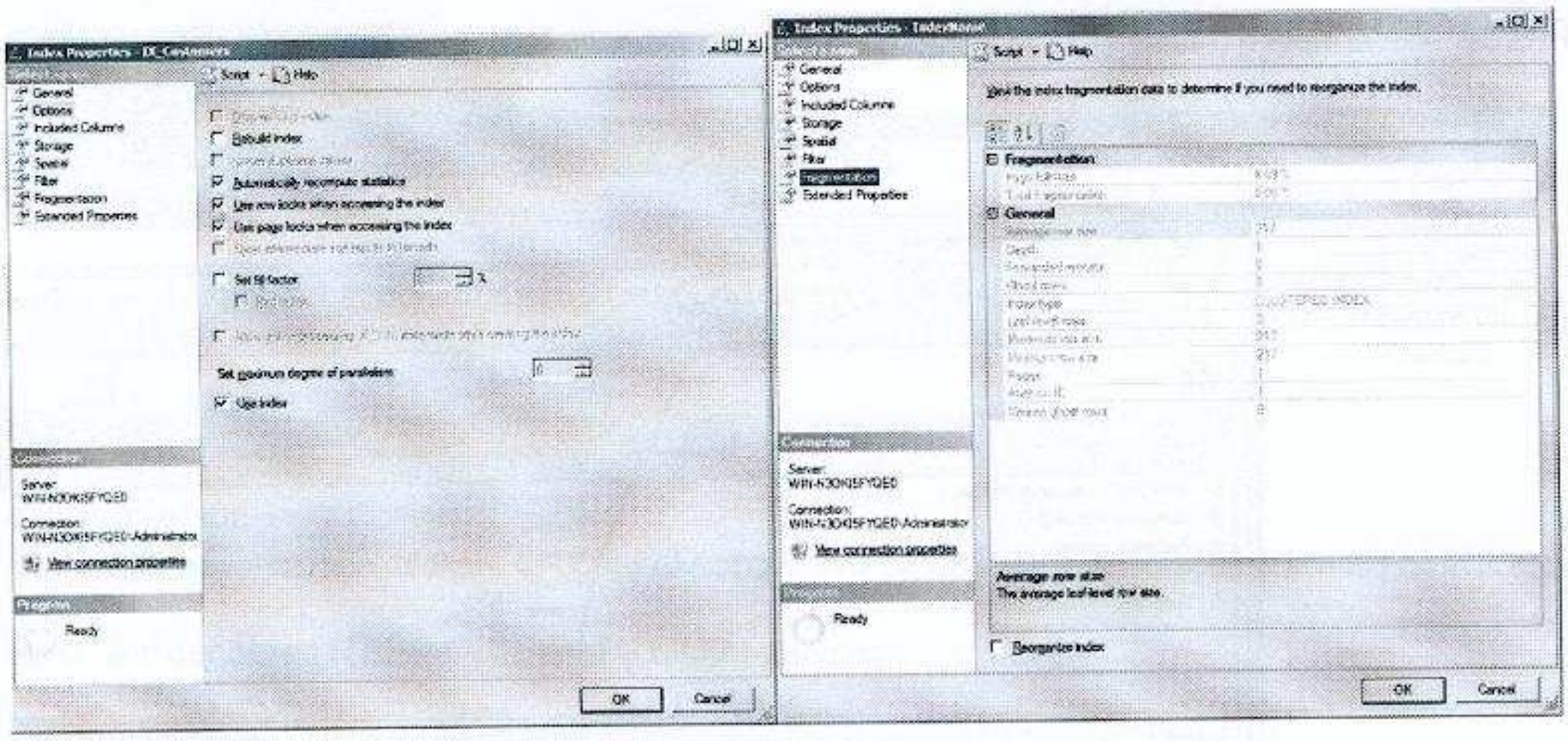
```
Query 2: Query cost (relative to the batch): 100%
select * from dbo.customers order by name
```

```
SELECT
Cost: 0 %
        Clustered Index Scan (Clustered)
        [Customers].[IndexName]
        Cost: 100 %
```

Er dit index fragmeteret tilstrækkeligt kan du enten rebuilde eller reorganisere det.

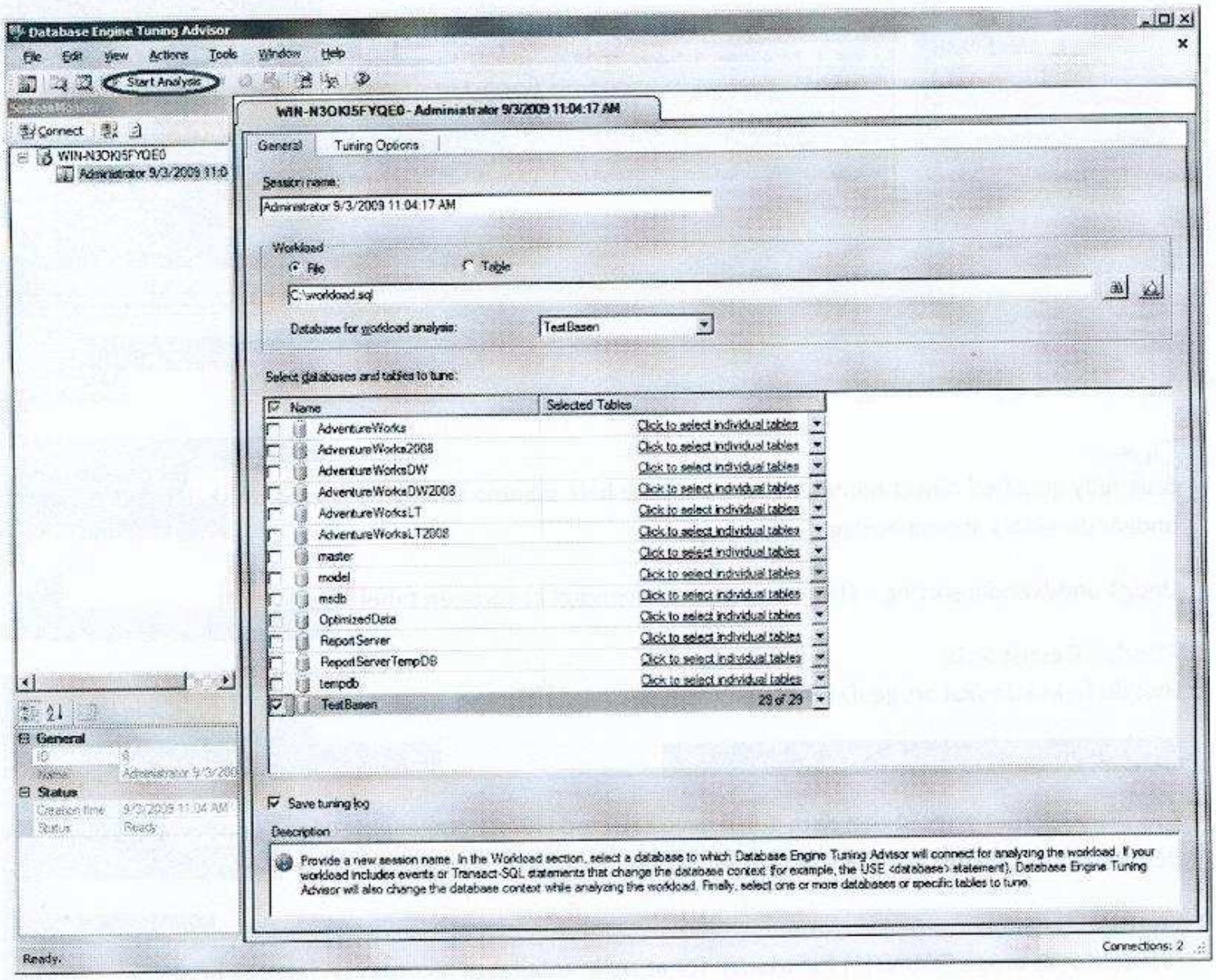
5 – 30 % = reorganisér

30%+ = Rebuild

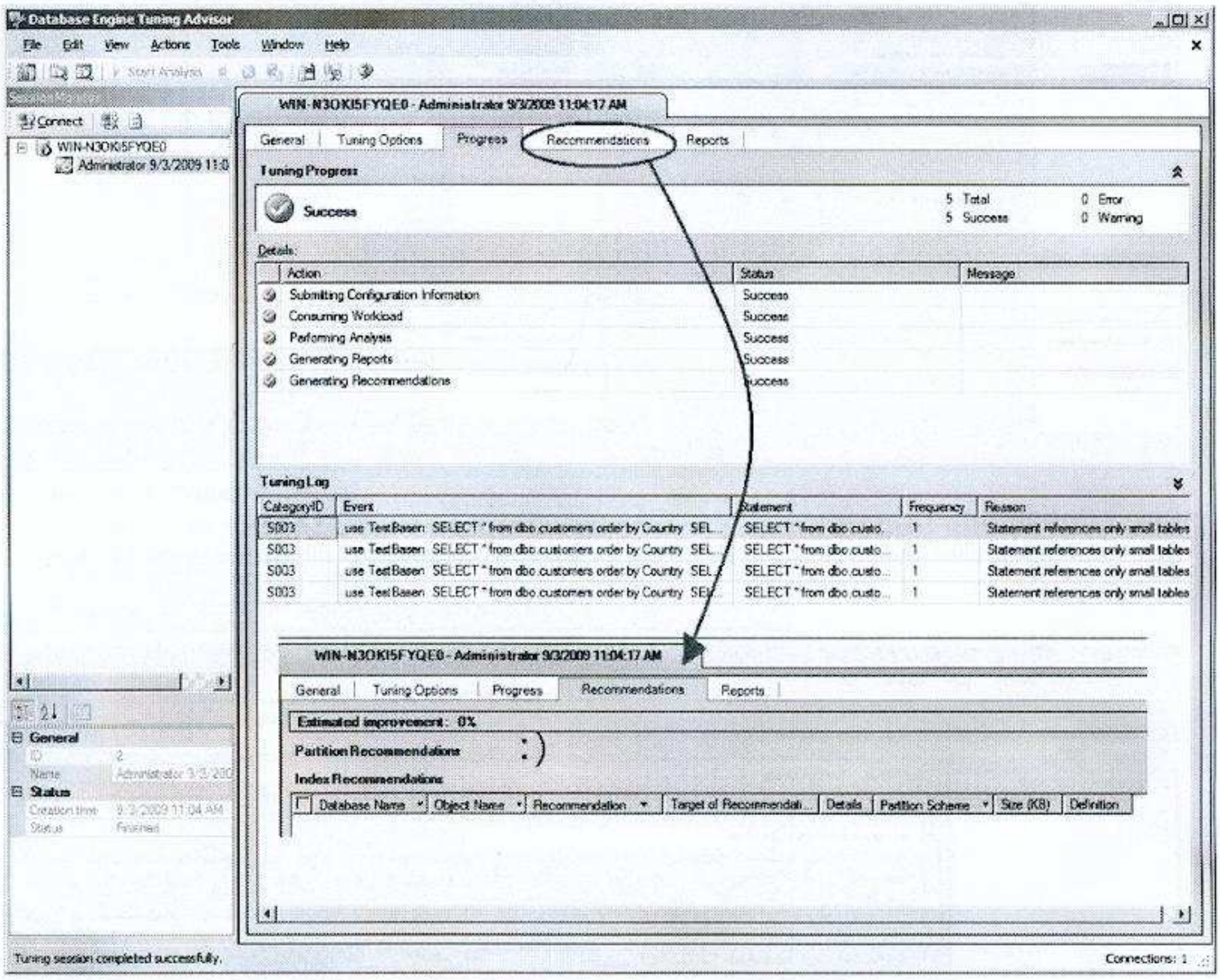


Query Optimizer

Query optimizer, eller engine tuning advisor, kan foreslå evt. ændringer til din database der kan øge performance:



Hvis der ikke er nogen forslag er alt jo godt:



SQL Kode Optimering

Tips

Brug fully qualified object names – Hvis du skriver hele schema navnet hver gang du skal have fat i noget undgår du ekstra shema opslag.

Undgå unødvendig sorting – Hvis det ikke er nødvendigt at sorte en tabel så lad være!

Mindre Result Sets

Hvis du f.eks kun skal bruge ID feltet i en tabel så undgå at bruge

```
Select * FROM %schemanavn%.%tabelnavn%
```

Specificér i stedet præcist den data du vil have selected i første omgang og undgå unødvendige rowcounts.
Definér altid så mange parametrer som muligt for at mindske antallet af opslag.

Counts

I stedet for at bruge COUNT(*) funktionen for at tælle antallet af records i en tabel så brug:

```
Select sum(row_count) 'TotalRows'
FROM sys.dm_db_partition_stats
Where object_id=object('%schemanavn%.%tabelnavn%')
and index_id<=1
```

Dette forgår så hutigt at den ofte vil registrere tidsforbruget som 0ms!

Resource Gouvenor

Konfigurering

Resource Gouvenor kan allokere CPU og RAM procentvis til forskellige user account groups på SQL'en.

Først skal der oprettes brugere og de skal tilføjes grupper:

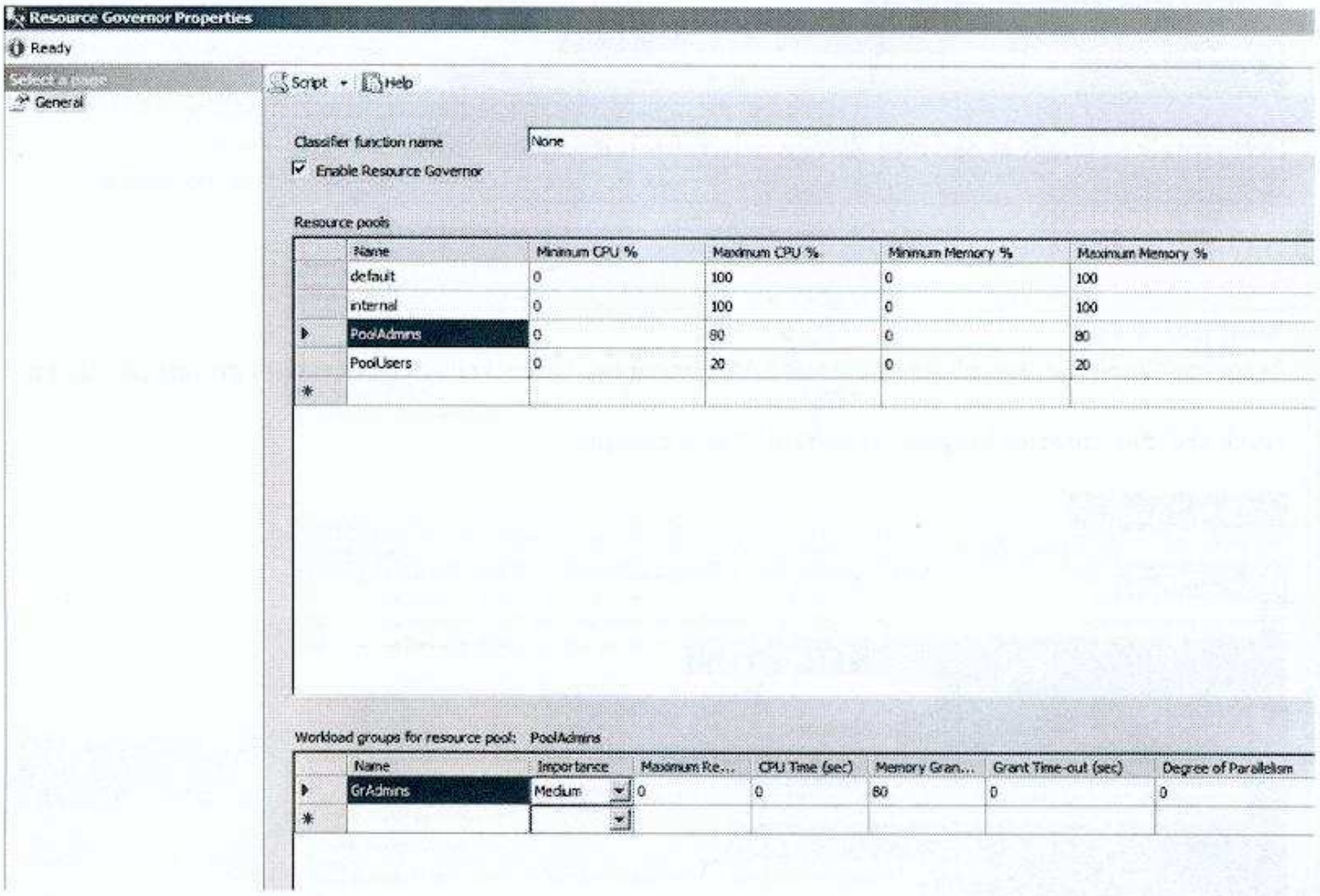
```
--Opret Brugere--

use testbasen
go
create login Admini with password='Admin1234',
default_database=testbasen
exec sp_grantdbaccess 'Admini'
exec sp_addrolemember 'db_datareader', 'Admini'
GO
create login Useri with password='User1234',
default_database=testbasen
exec sp_grantdbaccess 'Useri'
exec sp_addrolemember 'db_datareader', 'Useri'
GO
```

```
--Opret rettigheder --
Use master
go
create function fnClassifier()
returns sysname
with schemabinding
as
begin
declare @wrkGroup sysname

if suser_sname()='Admini' begin
set @wrkgroup='GrAdmins'
end else if suser_sname()='Useri' begin
set @wrkgroup='GrUsers'
end else begin
set @wrkgroup='default'
end
return @wrkgroup
end
go
```

Dernæst skal du konfigurere Guvernøren.



Og det var det!

Hvis en bruger nu f.eks er i gruppen GRAdmins (Ligesom Admini er – se koden) så har han hele 80% af CPU og Memory til rådighed!