**Analisando a Performance do Servidor - CheckList**  
Postado em Terça-feira, 5 de Abril de 2005 (11:01:58) por [niltonpinheiro](http://www.mcdbabrasil.com.br)

Uma das grandes dificuldades de  muitos DBA's, principalmente aqueles que estão iniciando no SQL Server, é saber como e o quê verificar no momento de uma lentidão no sistema ou até mesmo no servidor como um todo. Neste artigo, descreverei um checklist básico que você poderá seguir para ajudar na hora de identificar a causa de uma lentidão.

Na verdade, a causa da lentidão em um servidor pode ser várias: uma query que não faz uso adequado dos índices, tabelas sem índices, proplemas de modelagem, falta de memória, discos ruins e assim vai...

Independente da causa da lentidão, alguns passos básicos podem ser seguidos para tentar identificar o problema. Neste artigo estarei descrevendo alguns desses passos e dizendo como interpretá-los durante a verificação de problemas de performance em seu servidor. Porém, quero deixar claro que os passos decritos aqui não é um padrão e que a metodologia de análise pode variar de DBA para DBA.

**O PASSO-A-PASSO**

Passo 1: Verificar as conexões existentes no servidor e se existe problemas de blocks.

O primeiro passo para identificar problemas de performance é saber o que está rolando no servidor. Quais são as conexões existentes? O que elas estão executando? Existem conexões bloqueadas? Quem é o causador do bloqueio? Quem está bloqueado? etc...

A procedure de sistema sp\_who é uma procedure muito utilizada para responder estas questões, mas a principal pergunta a ser respondida aqui é: Existem conexões bloqueadas?

Para responder a esta pergunta execute a procedure sp\_who e verifique se existe alguma conexão onde a coluna Blk seja diferente de 0 (blk<>0). Se existir, já sabemos que estamos tendo problemas de block de conexões.

/\* SP\_WHO2 - improved \*/

declare @tempTable table (SPID INT,Status VARCHAR(255),Login VARCHAR(255),HostName VARCHAR(255),BlkBy VARCHAR(255),DBName VARCHAR(255),Command VARCHAR(255),CPUTime INT,DiskIO INT,LastBatch VARCHAR(255),ProgramName VARCHAR(255),SPID2 INT,REQUESTID INT);

INSERT INTO @tempTable

EXEC sp\_who2

select \* from @tempTable ORDER BY [@tempTable].DBName

Passo 1.2: Havendo conexões bloqueadas, identificar o causador do bloqueio e a instrução SQL envolvida.  
   
Uma vez detectado problemas de bloqueio, o próximo passo é identificar a conexão que está causando o bloqueio. Os bloqueios ocorrem porque normalmente duas ou mais conexões estão tentando utilizar um mesmo recurso. Imagine que temos duas conexões, uma com o SPID 64 e outra com o SPID 65. O SPID 65 efetua um lock exclusivo em uma tabela para efetuar uma atualização de dados e neste mesmo momento a conexão com SPID 64 tenta ler estas informações.

Neste caso, como a conexão SPID 65 abriu uma transação e efetuou um lock exclusivo, a conexão com SPID 64 ficará bloqueada pelo SPID 65 até que o mesmo conclua sua atualização. Uma das maiores causas de problemas de bloqueio são tabelas sem índices ou queries que não utilizam os índices de forma correta. Na maioria das vezes, a simples criação de uma índice clustered na tabela envolvida pode solucionar este tipo de problema.

Em grandes ambientes (ambientes com muitas conexões), a dificuldade aqui pode estar em identificar a conexão causadora do bloqueio. Isto porque muitas das vezes pode haver um certo encadeamento de bloqueio. Para facilitar este trabalho, você poderá estar utilizando a procedure [sp\_usrheadblocker](http://www.mcdbabrasil.com.br/modules.php?name=Scripts_Sql&op=search&query=sp_usrheadblocker).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* sp\_usrheadblocker \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if exists (select \* from dbo.sysobjects where id = object\_id(N'[dbo].[sp\_usrheadblocker]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)

drop procedure [dbo].[sp\_usrheadblocker]

GO

Create Proc dbo.sp\_usrheadblocker

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Procedure: sp\_usrheadblocker

Descrição:Possui a característica de reunir em apenas um comando as instruções: sp\_who Active

e DBCC InputBuffer(n). Retorna apenas as linhas que estão com Status de Blocked <> 0 e no

final de cada linha traz o InputBuffer.

A primeira linha sempre será a ponta (head blocker-a conexão que está causando o bloqueio de

todas as outras) e as demais linhas, as que estão aguardando a liberação do recurso.

Pode-se ver a quanto tempo cada uma das conexões está aguardando a liberação.

Autor Original: Marcelo Andretto

Adaptada por : Nilton Pinheiro

WebSite: http://www.mcdbabrasil.com.br

-- Melhor visualizada em modo ..GRADE..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

as

Set NoCount on

-- Tabelas de Apoio.

CREATE TABLE #tbheadBlocked (

[Host\_Id] [int] NULL ,

[SPID] [int] NULL ,

[a] [varchar] (14) NULL ,

[b] [int] NULL ,

[TextBuffer] [varchar] (255) NULL

)

Create Table #tbInputBuffer(a VarChar(14), b int, TextBuffer VarChar(255))

-- Passo1 -- Insere na tabela temporária "#tbheadblocked" todos os SPID que estão bloqueados

Insert Into #tbheadBlocked Select Host\_id(),SPID,null,null,null

from master..sysprocesses (NoLock)

Where SPID in (Select Blocked from master..sysprocesses (NoLock)

Where Blocked <>0)or Blocked <> 0

-- Passo 2 -- Abre um cursor para obter o DBCC InputBuffer de todas os SPID que foram inseridas

-- na tabela do passo 1 e armazena em uma nova tabela temporária "#tbInputBuffer"

Declare @SPID Int

Declare C\_Buffer CURSOR For Select SPID from #tbheadBlocked

Open C\_Buffer

Fetch C\_Buffer Into @SPID

While @@Fetch\_Status = 0

Begin

Insert Into #tbInputBuffer exec ('Dbcc InputBuffer(' + @SPID + ') with NO\_INFOMSGS ')

Update #tbheadBlocked Set TextBuffer = #tbInputBuffer.TextBuffer from #tbInputBuffer Where Spid = @spid

Fetch C\_Buffer Into @SPID

End

Close C\_Buffer

Deallocate C\_Buffer

-- Passo 3 -- Faz o Join das tabelas temporárias apresentando o resultado final.

-- Uma concatenação da SP\_Who Active + DBCC InputBuffer()

select distinct a.SPID, a.Blocked, a.ECID, a.WaitTime as WaitTimeMS,

datediff (mi,a.last\_batch,getDate() ) as RunAs, --Tempo de execução em minutos

SubString(a.Status,1,10) as Status, a.CPU,

SubString(Cast(a.Physical\_IO as Varchar(10)),1,10) as Physical\_IO,

SubString(a.HostName,1,15) as HostName,

SubString(a.LogiName,1,15) as LoginName,

SubString(DB\_Name(a.dbid),1,13) as DBName,

SubString(convert(VarChar(24),a.last\_batch ,113),1,24) as Last\_Batch,

a.open\_tran, a.MemUsage, b.TextBuffer

from master..sysprocesses a (NoLock) Right Outer Join #tbheadBlocked b (NoLock) On a.Spid = b.spid

Where a.SPID in (Select c.Blocked from master..sysprocesses c (NoLock)Where Blocked <>0) or a.Blocked <> 0

Order By a.Blocked

Set NoCount off

Drop Table #tbInputBuffer

Drop table #tbheadBlocked

GO

Ao executá-la você obterá as seguintes informações:

--**SPID** => Este é o identificador da conexão.  
--**Blocked** => informa o SPID que está bloqueando a conexão. Os SPIDs que possuirem Blocked=0 é o SPID que está causando os problemas de bloqueio.  
--**ECID** => Informa se a conexão esta fazendo uso de pararelismo.  
--**WaitTimeMS** => informa o tempo (em milessegundo) que a conexão está aguardando pelo recurso. Ou seja, a quanto tempo ela está bloqueada.  
--**RunAs** => informa a quanto tempo (em minutos) a conexão está no servidor.  
--**Status** => O status da conexão.  
--**CPU** => informa o consumo de CPU pela conexão.  
--**Physical\_IO** => informa o consumo de I/O pela conexão.  
--**HostName** => O nome da máquina que estabeleceu a conexão.  
--**LoginName** => o nome do login que estabeleceu a conexão.  
--**DBName** => o nome da base de dados.  
--**Last\_Batch** => a hora em que a conexão foi estabelecida ou executou a última instrução.  
--**open\_tran** => informa se existe alguma transação aberta pela conexão. O número apresentado aqui é o número de transações abertas pela conexão.  
--**MemUsage** => quantidade de memória sendo utilizada pela conexão.  
--**TextBuffer** => informa a instrução T-SQL sendo executada pela conexão.  
   
    
Na maioria das vezes, a primeira linha sempre será a conexão causadora do bloqueio e as demais, as que estão aguardando pela liberação do recurso. Nos casos onde você tiver block em vários recursos, as conexões causadoras dos boqueios sempre terão a coluna Blocked=0 e serão as primeiras linhas.

No exemplo abaixo, temos que as conexões 64 e 68 estão sendo bloqueadas pelas conexões 65 e 66 respectivamente. Na coluna TextBuffer, podemos ver que as conexões 65 e 64 estão utilizando o mesmo recurso (a tabela Products). O mesmo acontece com as conexões 66 e 68.

SPID   Blocked WaitTimeMS  TextBuffer  
------ ------- ----------- -----------------------------------------------------------------------  
65     0       0           begin tran Update products SET UnitsInStock= 38 where ProductID=1  
66     0       0           begin tran Update Orders SET ShipCountry = 'Brasil' where OrderID=10248  
64     65      1408218     select \* from Products  
68     66      134890      select \* from orders

Obs: Algumas colunas foram excluídas para facilitar a visualização.

Neste caso, as conexões 64 e 68 não serão liberadas enquanto as conexões 65 e 66 não concluírem o processo de atualização (efetuarem um commit tran) ou forem derrubadas com a utilização do comando Kill. Um ponto importante a ser observado é que nem sempre o comando Kill finalizará o processo de forma imediata. Dependendo da instrução sendo executada e dos status das conexões, a conexão pode entrar em Rollback e demorar um certo tempo para ser finalizada.

Caso a coluna TextBuffer não esteja lhe fornecendo informações suficientes sobre a instrução T-SQL sendo executada pela conexão, você pode executar também o comando DBCC Inputbuffer(<spid>). Porém, uma limitação do DBCC Inputbuffer é que nos casos em que a conexão estiver executando uma stored procedure ou view, ele lhe mostrará apenas o nome da procedure ou view. O ideal seria poder identificar o que realmente está sendo executado pela conexão naquele exato momento e para vencer esta limitação do comando DBCC Inputbuffer, você pode utilizar a procedure [sp\_usrinputbuffer](http://www.mcdbabrasil.com.br/modules.php?name=Scripts_Sql&op=search&query=sp_usrinputbuffer)<spid>.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* sp\_usrinputbuffer \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

-- Procedure para identificar o ponto exato de execução de um statement

USE MASTER

GO

if exists (select \* from dbo.sysobjects where id = object\_id(N'[dbo].[sp\_usrinputbuffer]') and OBJECTPROPERTY(id, N'IsProcedure') = 1)

drop procedure [dbo].[sp\_usrinputbuffer]

GO

CREATE PROCEDURE dbo.sp\_usrinputbuffer

(

@SPID smallint, -- O SPID a ser analizado

@WAIT tinyint = 1, -- Intervalo de repetição entre as verificações.Valor deve estar entre 0 e 60 segundos

-- Default é de 1 em 1 segundo

@NoLoop bit = 1 -- Se =1, o SPID é analizado apenas uma vez. Se =0 fica em loop até o final do processo.

)

AS

BEGIN

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sp\_usrinputbuffer: Esta procedure permite identificar a instrução exata que está sendo

executado por uma dada conexão. Semelhante ao DBCC INPUTBUFFER mas diferente desta, em caso

de procedure, a proc mostra toda a instrução inteira e não apenas o nome da procedure. Muito boa

para ser usada em conjunto com a procedure sp\_usrheadblocker.

Exemplo: sp\_usrinputbuffer 54 (Onde 54 é o ID da conexão SPID)

Obs: Se preferir, esta pode ser criada em qualquer banco, porém, ao executá-la deverá passar o nome da

base onde a mesma se encontra. Exemplo: Base..sp\_usrinputbuffer 54

Author Original: Narayana Vyas Kondreddi

Source: http://vyaskn.tripod.com

Date Created: 18/12/2003

Alterada por : Nilton Pinheiro

WebSite: http://www.mcdbabrasil.com.br

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SET NOCOUNT ON

DECLARE @sql\_handle binary(20), @handle\_found bit

DECLARE @stmt\_start int, @stmt\_end int

DECLARE @line varchar(8000), @wait\_str varchar(8)

SET @handle\_found = 0

IF @WAIT NOT BETWEEN 0 AND 60

BEGIN

RAISERROR('Valores válidos para @WAIT estão entre 0 e 60 segundos', 16, 1)

RETURN -1

END

ELSE

BEGIN

SET @wait\_str = '00:00:' + RIGHT('00' + CAST(@WAIT AS varchar(2)), 2)

END

WHILE 1 = 1

BEGIN

SELECT @sql\_handle = sql\_handle,

@stmt\_start = stmt\_start/2,

@stmt\_end = CASE WHEN stmt\_end = -1 THEN -1 ELSE stmt\_end/2 END

FROM master.dbo.sysprocesses

WHERE spid = @SPID

AND ecid = 0

IF @sql\_handle = 0x0

BEGIN

IF @handle\_found = 0

BEGIN

RAISERROR('Não pode encontrar o handle ou o SPID é inválido', 16, 1)

RETURN -1

END

ELSE

BEGIN

RAISERROR('Query/Stored procedure completada', 0, 1)

RETURN 0

END

END

ELSE

BEGIN

SET @handle\_found = 1

END

Print '\*\*\*\*\*\*\*\* STATEMENT SENDO EXECUTADO NO MOMENTO \*\*\*\*\*\*\*\*\*\*\*\*'

Print ''

SET @line =

(

SELECT

SUBSTRING( text,

COALESCE(NULLIF(@stmt\_start, 0), 1),

CASE @stmt\_end

WHEN -1

THEN DATALENGTH(text)

ELSE

(@stmt\_end - @stmt\_start)

END

)

FROM ::fn\_get\_sql(@sql\_handle)

)

Print @line

IF @NoLoop = 1

BEGIN

RETURN 0

END

WAITFOR DELAY @wait\_str

END

END

GO

Exemplo:

-- Executa o Comando DBCC InputBuffer no QA  
dbcc inputbuffer (77)

EventType      Parameters EventInfo                            
-------------- ---------- ----------------------------------   
Language Event 0          exec SalesByCategory 'Beverages'

-- Para usar esta procedure o conexão deve estar com o status "Runnable"  
Exec sp\_usrinputbuffer 77

\*\*\*\*\*\*\*\* STATEMENT SENDO EXECUTADO NO MOMENTO \*\*\*\*\*\*\*\*\*\*\*\*  
SELECT ProductName,TotalPurchase=ROUND(SUM(CONVERT(decimal(14,2), OD.Quantity \* (1-OD.Discount) \* OD.UnitPrice)), 0)  
FROM [Order Details] OD, Orders O, Products P, Categories C  
WHERE OD.OrderID = O.OrderID   
AND OD.ProductID = P.ProductID   
AND P.CategoryID = C.CategoryID  
AND C.CategoryName = @CategoryName  
AND SUBSTRING(CONVERT(nvarchar(22), O.OrderDate, 111), 1, 4) = @OrdYear  
GROUP BY ProductName  
ORDER BY ProductName

Passo 2: Avaliar o estado do servidor.

Se o problema de lentidão ou performance não estiver associado com o bloqueio das conexões, o próximo passo a seguir é fazer uma análise em alguns contadores do **Perfmon (Performance Monitor)** para tentar identificar possíveis gargalos no nível de hardware.

Alguns dos principais contadores a serem monitorados são listados abaixo:

A) Contadores de Memória  
   
**Memory: Available Bytes** ==>  Indica a quantidade de memória disponível no servidor.

**Memory: Pages/Sec**   ==> Indica se está havendo paginação. Em perfeitas condições deve-se ver pouca atividade neste contador. Se o SQL esta rodando em um servidor dedicado, toda a sua memória alocada (aquela configurada nas propriedades do servidor dentro do Enterprise Manager) será mapeada para a memória física e muito pouco swapping (paginação) deverá ocorrer. Se este contador estiver muito alto ou crescente, verifique a memória alocada para o SQL Server, aumentar este valor ou colocar o SQL Server para configurar sua memória dinamicamene pode ajudar a reduzir a paginação.

B) Contadores do SQL Server  
   
**SQL Server: Buffer Manager: Buffer Cache Hit Ratio** ==> Indica o percentual de requisições a dados que são obtidas no próprio cache do sql e portanto, sem precisar acessar o disco. O ideal é estar maior que 90%.

**SQL Server:Access Methods:Page Splits/sec** ==> Mostra quantos page splits estão ocorrendo no servidor. Este valor deve ser o mais baixo possível. Se o valor estiver alto, configurar os índices com um fillfactor apropriado pode ajudar a reduzir este valor.

**SQL Server: Memory Manager: Total Server Memory (KB)** ==> Quando maior que a quantidade de memória física pode indicar falta de memória.

 C) Contadores de CPU  
   
**Processor:% Processor Time:\_Total** ==> Indica o consumo de CPU no servidor (ideal abaixo de 80%).

**Process: % Processor time: sqlservr** ==> Indica o consumo do processador pelo processo do SQL Server.

**System: Processor Queue Length** ==> Indica o número de threads aguardando para execução no processador e nunca deve exceder 1 ou 2 (por processador) por um período superior a 10 minutos.

D) Contadores de Disco

**PhysicalDisk: % idle Time** ==> Indica o percentual de tempo que o disco esta ocioso. Supostamente este contador apresenta uma estimativa mais precisa sobre a utilização dos discos. Subtraindo o valor encontrado de 100, temos uma ideia do quanto o disco esta trabalhando.

**PhysicalDisk: % Disk Time** ==> Mostra o quanto o disco esta ocupado. Se valor > 60% durante 10 minutos, verificar os contadores **PhysicalDisk: % Disk Read Time** e **PhysicalDisk: % Disk Write Time** para identificar se é leitura ou escrita quem está causando maior utilização.

**PhysicalDisk: Avg. Disk Queue Length** ==> Mostra o número médio de requisições de I/O aguardando para acesso ao disco. Este contador nunca deve exceder 2 (por disco) por um extenso período de tempo.

**PhysicalDisk: Avg. Disk Sec/Transfer** ==> Taxa de transferência em bytes de ou para um disco durante operações de leitura ou escrita.

D) Contadores de Banco de Dados

**SQL Server: Access Methods: Full Scans/sec** ==> Mostra a quantidade de Table Scan sendo executada no SQL Server. Este valor deve ser o mais baixo possível e se for constantemente alto, use o Profile para identificar quais queries estão causando os scans.

**SQL Server: Locks: Average Wait Time (ms)** ==> Este contador mostra quanto tempo em milessegundos um processo está aguardando para obter lock no SQL Server. O valor ideal para este contador é zero. Se este valor estiver muito alto, utilize o Profile para identificar quais processos estão mantendo locks por um período longo de tempo.

Como descrito no início deste artigo, o problema de permance pode estar associado a várias causas, este checklist segue alguns passos básicos para ajudar na identificação do problema. Também é importante observar que este checklist não é um padrão e que por isso, os métodos utilizados para analizar a performance de um servidor pode variar de DBA para DBA. Na verdade, no dia-a-dia de trabalho cada DBA acaba por criar a sua própria metodologia de análise.

Um abraço a todos  
Nilton Pinheiro

Fonte: http://www.mcdbabrasil.com.br/modules.php?name=News&file=article&sid=170

USE Master

GO

EXEC sp\_who2

GO

sp\_who

sp\_who active

### sys.dm\_exec\_requests DMV

USE Master

GO

SELECT \* FROM sys.dm\_exec\_requests WHERE blocking\_session\_id <> 0;

GO

### sys.dm\_os\_waiting\_tasks DMV

USE Master

GO

SELECT session\_id, wait\_duration\_ms, wait\_type, blocking\_session\_id FROM sys.dm\_os\_waiting\_tasks WHERE blocking\_session\_id <> 0

GO

# How to identify blocking in SQL Server 2005 and 2008

|  |  |
| --- | --- |
|  | By: [Nitansh Agarwal](http://www.mssqltips.com/sqlserverauthor/61/nitansh-agarwal/)   |   [Read Comments (23)](http://www.mssqltips.com/sqlservertip/2429/how-to-identify-blocking-in-sql-server-2005-and-2008/#comments)   |   Related Tips: [More](http://www.mssqltips.com/sql_server_dba_tips.asp) > [Locking and Blocking](http://www.mssqltips.com/sql-server-tip-category/61/locking-and-blocking/) |

##### http://www.mssqltips.com/sqlservertip/2429/how-to-identify-blocking-in-sql-server-2005-and-2008/

##### Problem

In our SQL Server environment, we have frequent [locking](http://www.mssqltips.com/sqlservertip/1968/understanding-sql-server-locking/) and [blocking](http://www.mssqltips.com/sqlservertip/1978/understanding-sql-server-blocking/) across a few different versions of SQL Server. How can I find blocking and blocked SPID's in SQL Server 2005 and later versions? Is there only one way to find out which spids are blocking? Are there any commands that I can run against multiple SQL Server versions? Check out this tip to learn more about locking and blocking.

##### Solution

Whenever a user contacts the DBA team indicating a processes looks hung or a process is not proceeding checking the applicable database blocking makes a great deal of sense. Blocking happens when one connection from an application holds a lock and a second connection requires a conflicting lock. This forces the second connection to be blocked until the first connection completes. With this being said, locking is a natural occurrence in SQL Server in order to maintain data integrity. For more information about locking and blocking review these tips: [Understanding SQL Server Locking](http://www.mssqltips.com/sqlservertip/1968/understanding-sql-server-locking/) and[Understanding SQL Server Blocking](http://www.mssqltips.com/sqlservertip/1978/understanding-sql-server-blocking/).

There are number of ways to find out the details of the system processes IDs (spids) involved in blocking. I have tried to cover some of the options in this tip to include:

* sp\_who2 System Stored Procedure
* sys.dm\_exec\_requests DMV
* Sys.dm\_os\_waiting\_tasks
* SQL Server Management Studio Activity Monitor
* SQL Server Management Studio Reports
* SQL Server Profiler

### sp\_who2 System Stored Procedure

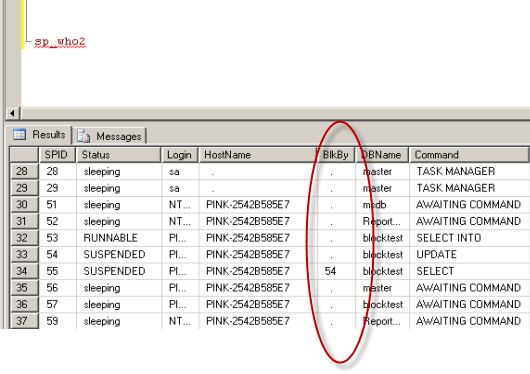
The sp\_who2 system stored procedure provides information about the current SQL Server processes with the associated users, application, database, CPU time, etc. The information returned can be filtered to return only the active processes by using the ‘active' parameter. Below is some sample code and a screen shot with showing process 55 being blocked by process 54.

USE Master

GO

EXEC sp\_who2

GO



Additional resources:

* [SQL Server Command Line Tools To Manage Your Server](http://www.mssqltips.com/sqlservertip/1029/sql-server-command-line-tools-to-manage-your-server/)
* [Locking and Blocking Scripts in SQL Server 2000 vs SQL Server 2005](http://www.mssqltips.com/sqlservertip/1359/locking-and-blocking-scripts-in-sql-server-2000-vs-sql-server-2005/)
* [Killing a SPID and Checking Rollback progress in SQL Server](http://www.mssqltips.com/sqlservertip/1473/killing-a-spid-and-checking-rollback-progress-in-sql-server/)

### sys.dm\_exec\_requests DMV

The sys.dm\_exec\_requests DMV provides details on all of the processes running in SQL Server. With the WHERE condition listed below, only blocked processes will be returned.

USE Master

GO

SELECT \*

FROM sys.dm\_exec\_requests

WHERE blocking\_session\_id <> 0;

GO

Additional resources:

* [How to isolate the current running commands in SQL Server](http://www.mssqltips.com/sqlservertip/1811/how-to-isolate-the-current-running-commands-in-sql-server/)
* [Identifying the input buffer in SQL Server 2000 vs SQL Server 2005](http://www.mssqltips.com/sqlservertip/1363/identifying-the-input-buffer-in-sql-server-2000-vs-sql-server-2005/)
* [Finding a SQL Server process percentage complete with dynamic management views](http://www.mssqltips.com/sqlservertip/1338/finding-a-sql-server-process-percentage-complete-with-dmvs/)

### sys.dm\_os\_waiting\_tasks DMV

The sys.dm\_os\_waiting\_tasks DMV returns information about the tasks that are waiting on resources. To view the data, users should have SQL Server System Administrator or VIEW SERVER STATE permissions on the instance.

USE Master

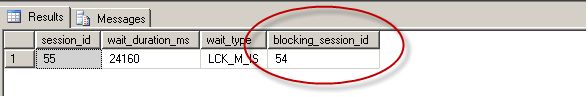
GO

SELECT session\_id, wait\_duration\_ms, wait\_type, blocking\_session\_id

FROM sys.dm\_os\_waiting\_tasks

WHERE blocking\_session\_id <> 0

GO

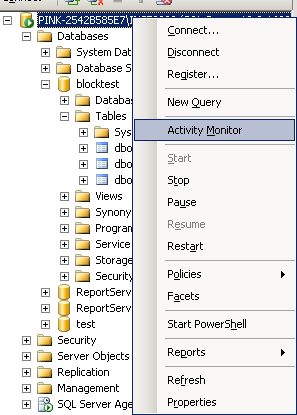


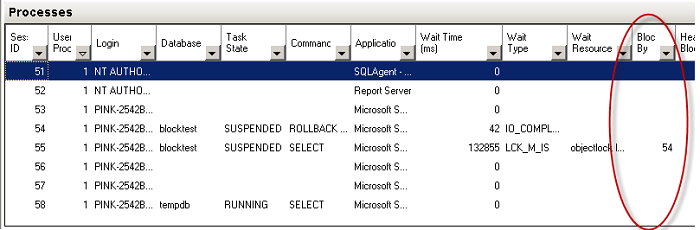
Additional resources:

* [SQL Server Tempdb Usage and Bottlenecks tracked with Extended Events](http://www.mssqltips.com/sqlservertip/1853/sql-server-tempdb-usage-and-bottlenecks-tracked-with-extended-events/)

### SQL Server Management Studio Activity Monitor

If you are more comfortable using [SQL Server Management Studio](http://www.mssqltips.com/category.asp?catid=52) to review locking and blocking as opposed to querying system objects or executing stored procedures, you are in luck. There are even a few different tools in SQL Server Management Studio you can use. The first option is the **Activity Monitor**, which can be accessed by navigating to the instance name | right click | select 'Activity Monitor'. To view the Activity Monitor in SQL Server 2005 and SQL Server 2008, users should have SQL Server System Administrator or VIEW SERVER STATE permissions on the instance.



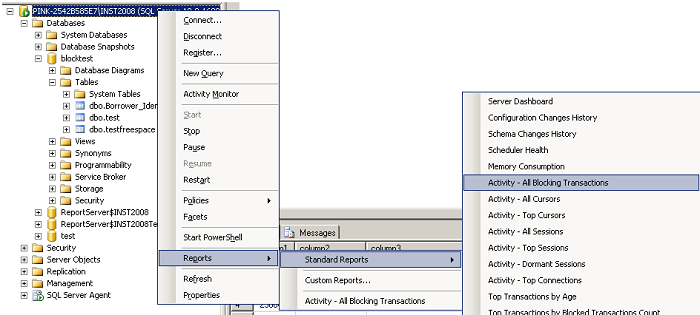


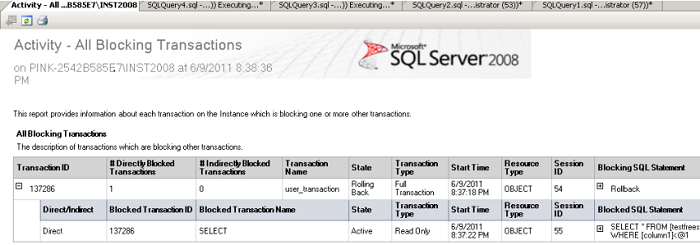
Additional resources:

* [Performance Analysis Using SQL Server 2008 Activity Monitor Tool](http://www.mssqltips.com/sqlservertip/1917/performance-analysis-using-sql-server-2008-activity-monitor-tool/)

### SQL Server Management Studio Reports

The second option in SQL Server Management Studio to monitor blocking is with the standard reports, which can be accessed by navigating to the instance name | right click | **Reports | Standard Reports | Activity - All Blocking Transactions**. Once again, users should have SQL Server System Administrator or VIEW SERVER STATE permissions on the instance.





Additional resources:

* [Built-In Performance Reports in SQL Server 2005](http://www.mssqltips.com/sqlservertip/1179/built-in-performance-reports-in-sql-server-2005/)

### SQL Server Profiler

To capture blocking related data on a continuous basis, one option is to run SQL Server Profiler and save the data to a table or file for analysis purposes. In order to configure Profiler to capture blocking related data, execute **Profiler**, configure the general properties then navigate to Event Selection tab | select Show all events | Errors and Warnings | check the Blocked process report and then run the application. In addition, be sure to configure the 'blocked process threshold' before you start Profiler using this code:

sp\_configure 'show advanced options', 1

GO

RECONFIGURE

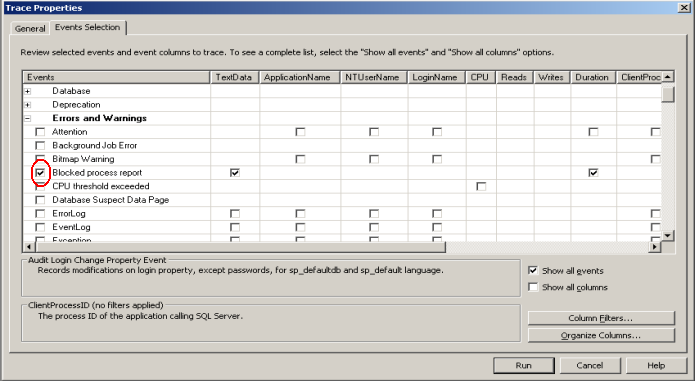
GO

sp\_configure 'blocked process threshold', 20

GO

RECONFIGURE

GO



Additional resources:

* Tip category: [Profiler and Trace](http://www.mssqltips.com/category.asp?catid=83)
* [Capturing SQL Server Deadlock Information in XML Format](http://www.mssqltips.com/sqlservertip/1234/capturing-sql-server-deadlock-information-in-xml-format/)
* [Finding and troubleshooting SQL Server deadlocks](http://www.mssqltips.com/sqlservertip/1036/finding-and-troubleshooting-sql-server-deadlocks/)
* [SQL Server Profiler Graphical Deadlock Chain](http://www.mssqltips.com/sqlservertip/1222/sql-server-profiler-graphical-deadlock-chain/)

##### Next Steps

* If you are faced with a blocking situation, be sure to consider all of your options in the short and long term. To resolve the immediate issue, you may need to KILL some spids, but to resolve the issue you may need to change your database design, change your data access, add NOLOCK hints to particular queries, etc.
* Check out these tips to learn more about locking and blocking
  + [Understanding SQL Server Locking](http://www.mssqltips.com/sqlservertip/1968/understanding-sql-server-locking/)
  + [Understanding SQL Server Blocking](http://www.mssqltips.com/sqlservertip/1978/understanding-sql-server-blocking/)
  + Tip category: [Locking and Blocking Tips](http://mssqltips.com/category.asp?catid=61)

Last Update: 7/15/2011

# INF: Understanding and resolving SQL Server blocking problems

http://support.microsoft.com/kb/224453

[Article translations [Article translations](javascript:void(0);)](javascript:void(0);)

Article ID: 224453 - [View products that this article applies to.](http://support.microsoft.com/kb/224453#appliesto)

[Expand all](javascript:void(0);) | [Collapse all](javascript:void(0);)

## [Summary](javascript:void(0);)

In this article, the term "connection" refers to a single logged-on session of the database. Each connection appears as a Session ID (SPID). Each of these SPIDs is often referred to as a process, although it is not a separate process context in the usual sense. Rather, each SPID consists of the server resources and data structures necessary to service the requests of a single connection from a given client. A single client application may have one or more connections. From the perspective of SQL Server, there is no difference between multiple connections from a single client application on a single client computer and multiple connections from multiple client applications or multiple client computers. One connection can block another connection, regardless of whether they emanate from the same application or separate applications on two different client computers.

## [More information](javascript:void(0);)

Blocking is an unavoidable characteristic of any relational database management system (RDBMS) with lock-based concurrency. On SQL Server, blocking occurs when one SPID holds a lock on a specific resource and a second SPID attempts to acquire a conflicting lock type on the same resource. Typically, the time frame for which the first SPID locks the resource is very small. When it releases the lock, the second connection is free to acquire its own lock on the resource and continue processing. This is normal behavior and may happen many times throughout the course of a day with no noticeable effect on system performance.  
  
The duration and transaction context of a query determine how long its locks are held and, thereby, their impact on other queries. If the query is not executed within a transaction (and no lock hints are used), the locks for SELECT statements will only be held on a resource at the time it is actually being read, not for the duration of the query. For INSERT, UPDATE, and DELETE statements, the locks are held for the duration of the query, both for data consistency and to allow the query to be rolled back if necessary.  
  
For queries executed within a transaction, the duration for which the locks are held are determined by the type of query, the transaction isolation level, and whether or not lock hints are used in the query. For a description of locking, lock hints, and transaction isolation levels, see the following topics in SQL Server Books Online:

* Locking in the Database Engine
* Customizing Locking and Row Versioning
* Lock Modes
* Lock Compatibility
* Row Versioning-based Isolation Levels in the Database Engine
* Controlling Transactions (Database Engine)

When locking and blocking increase to the point where there is a detrimental effect on system performance, it is usually due to one of the following reasons:

* A SPID holds locks on a set of resources for an extended period of time before releasing them. This type of blocking resolves itself over time, but can cause performance degradation.
* A SPID holds locks on a set of resources and never releases them. This type of blocking does not resolve itself and prevents access to the affected resources indefinitely.

In the first scenario above, the blocking problem resolves itself over time as the SPID releases the locks. However, the situation can be very fluid as different SPIDs cause blocking on different resources over time, creating a moving target. For this reason, these situations can be difficult to troubleshoot using SQL Server Enterprise Manager or individual SQL queries. The second situation results in a consistent state that can be easier to diagnose.

#### Gathering Blocking Information

To counteract the difficulty of troubleshooting blocking problems, a database administrator can use SQL scripts that constantly monitor the state of locking and blocking on SQL Server. These scripts can provide snapshots of specific instances over time, leading to an overall picture of the problem. For a description of how to monitor blocking with SQL scripts, see the following articles in the Microsoft Knowledge Base:

[271509](http://support.microsoft.com/kb/271509) How to monitor blocking in SQL Server 2005 and in SQL Server 2000

The scripts in this article will perform the tasks below. Where possible, the method for obtaining this information from SQL Server Management Studio is given.

1. **Identify the SPID (Session ID) at the head of the blocking chain and the SQL Statement.**  
   In addition to using the scripts in the previously mentioned Knowledge Base article, you can identify the head of the blocking chain by using features that are provided through SQL Server Management Studio. To do this, use one of the following methods:
   * Right-click the server object, expand **Reports**, expand **Standard Reports**, and then click **Activity – All Blocking Transactions**. This report shows the transactions at the head of blocking chain. If you expand the transaction, the report will show the transactions that are blocked by the head transaction. This report will also show the "Blocking SQL Statement" and the "Blocked SQL Statement."
   * Use **DBCC INPUTBUFFER(<spid>)** to find the last statement that was submitted by a SPID.
2. **Find the transaction nesting level and process status of the blocking SPID.**  
   The transaction nesting level of a SPID is available in the @@TRANCOUNT global variable. However, it can be determined from outside the SPID by querying the **sysprocesses** table as follows:
3. SELECT open\_tran FROM master.sys.sysprocesses WHERE SPID=<blocking SPID number>
4. go

The value returned is the @@TRANCOUNT value for the SPID. This shows the transaction nesting level for the blocking SPID, which in turn can explain why it is holding locks. For example, if the value is greater than zero, the SPID is in the midst of a transaction (in which case it is expected that it retains certain locks it has acquired, depending on the transaction isolation level).  
  
You can also check to see if any long-term open transaction exists in the database by using **DBCC OPENTRAN (database\_name)**.

#### Gathering SQL Server Profiler Trace Information

In addition to the above information, it is often necessary to capture a Profiler trace of the activities on the server to thoroughly investigate a blocking problem on SQL Server. If a SPID executes multiple statements within a transaction, only the last statement that was submitted will show in the report, input buffer, or activity monitor output. However, one of the earlier commands may be the reason locks are still being held. A Profiler trace will enable you to see all of the commands executed by a SPID within the current transaction. The following steps help you to set up SQL Server Profiler to capture a trace.

1. Open SQL Server Profiler.
2. On the **File** menu, point to **New**, and then click **Trace**.
3. On the **General** tab, specify a trace name and a file name to capture the data to.  
     
   **Important:** The trace file should be written to a fast local or shared disk. Avoid tracing to a slow disk or network drive. Also make sure Server processes trace data is selected.
4. On the **Events Selection** tab, click to select the **Show all events** and the **Show all columns** check boxes.
5. On the **Events Selection** tab, add the Event types that are listed in Table 1 to your trace.  
     
   Additionally, you may include the additional Event types that are listed in Table 2 for further information. If you are running in a high-volume production environment, you may decide to use only the events in Table 1, as they are typically sufficient to troubleshoot most blocking problems. Including the additional events in Table 2 may make it easier to quickly determine the source of a problem (or these events may be necessary to identify the culprit statement in a multi-statement procedure). However, including events in Table 2 will also add to the load on the system and increase the trace output size.

**Table 1: Event types**

|  |  |
| --- | --- |
| Heading | Event |
| Errors and Warnings | Exception |
| Errors and Warnings | Attention |
| Security Audit | Audit Login |
| Security Audit | Audit Logout |
| Sessions | Existing Connection |
| Stored Procedures | RPC:Starting |
| TSQL | SQL:BatchStarting |

**Table 2: Additional Event types**

|  |  |
| --- | --- |
| Heading | Event |
| Transactions | DTCTransaction |
| Transactions | SQLTransaction |
| Stored Procedures | RPC:Completed |
| TSQL | SQL:BatchCompleted |
| Stored Procedures | SP:StmtStarting |
| Stored Procedures | SP:StmtCompleted |

For more information about using the SQL Server Profiler, please see SQL Server Books Online.

### Identifying and Resolving Common Blocking Scenarios

By examining the above information, you can determine the cause of most blocking problems. The rest of this article is a discussion of how to use this information to identify and resolve some common blocking scenarios. This discussion assumes you have used the blocking scripts in article 271509 (referenced earlier) to capture information on the blocking SPIDs and have made a Profiler trace with the events described above.

#### Viewing the Blocking Script Output

##### Examine the sys.sysprocesses output to determine the heads of the blocking chains

If you did not specify fast mode for the blocking scripts, there will be a section titled "SPIDs at the head of blocking chains" that lists the SPIDs that are blocking other SPIDs in the script output.

SPIDs at the head of blocking chains

If you specified the fast option, you can still determine the blocking heads by looking at the **sys.sysprocesses** output and following the hierarchy of the SPID that is reported in the blocked column.

##### Examine the sys.sysprocesses output for information on the SPIDs at the head of the blocking chain.

It is important to evaluate the following **sys.sysprocesses** fields:  
  
**Status**  
This column shows the status of a particular SPID. Typically, a **sleeping** status indicates that the SPID has completed execution and is waiting for the application to submit another query or batch. A **runnable**, **running**, or **sos\_scheduler\_yield** status indicates that the SPID is currently processing a query. The following table gives brief explanations of the various status values.

|  |  |
| --- | --- |
| **Status** | **Meaning** |
| Background | The SPID is running a background task, such as deadlock detection. |
| Sleeping | The SPID is not currently executing. This usually indicates that the SPID is awaiting a command from the application. |
| Running | The SPID is currently running on a scheduler. |
| Runnable | The SPID is in the runnable queue of a scheduler and waiting to get scheduler time. |
| Sos\_scheduler\_yield | The SPID was running, but it has voluntarily yielded its time slice on the scheduler to allow another SPID to acquire scheduler time. |
| Suspended | The SPID is waiting for an event, such as a lock or a latch. |
| Rollback | The SPID is in rollback of a transaction. |
| Defwakeup | Indicates that the SPID is waiting for a resource that is in the process of being freed. The waitresource field should indicate the resource in question. |

**Open\_tran**  
  
This field tells you the transaction nesting level of the SPID. If this value is greater than 0, the SPID is within an open transaction and may be holding locks acquired by any statement within the transaction.  
  
**Lastwaittype, waittype, and waittime**  
  
The **lastwaittype** field is a string representation of the **waittype** field, which is a reserved internal binary column. If the **waittype** is 0x0000, the SPID is not currently waiting for anything and the **lastwaittype** value indicates the last **waittype** that the SPID had. If the **waittype** is not zero, the **lastwaittype** value indicates the current **waittype** of the SPID.   
  
For a brief description of the different **lastwaittype** and **waittype** values, see the following article in the Microsoft Knowledge base:

[822101](http://support.microsoft.com/kb/822101) Description of the waittype and lastwaittype columns in the master.dbo.sysprocesses table in SQL Server 2000 and SQL Server 2005

For more information about **sys.dm\_os\_wait\_stats**, see SQL Server Books Online.  
  
The **waittime** value can be used to determine if the SPID is making progress. When a query against the **sys.sysprocesses** table returns a value in the **waittime** column that is less than the **waittime** value from a previous query of **sys.sysprocesses**, this indicates that the prior lock was acquired and released and is now waiting on a new lock (assuming non-zero waittime). This can be verified by comparing the **waitresource** between **sys.sysprocesses** output.  
  
**Waitresource**  
  
This field indicates the resource that a SPID is waiting on. The following table lists common **waitresource** formats and their meaning:

|  |  |  |
| --- | --- | --- |
| Resource | Format | Example |
| Table | DatabaseID:ObjectID:IndexID | TAB: 5:261575970:1 In this case, database ID 5 is the **pubs** sample database and object ID 261575970 is the **titles** table and 1 is the clustered index. |
| Page | DatabaseID:FileID:PageID | PAGE: 5:1:104 In this case, database ID 5 is **pubs**, file ID 1 is the primary data file, and page 104 is a page belonging to the **titles** table.   To identify the object id that the page belongs to, use the DBCC PAGE (dbid, fileid, pageid, output\_option) command, and look at the m\_objId. For example:  DBCC TRACEON ( 3604 )  DBCC PAGE ( 5 , 1 , 104 , 3 ) |
| Key | DatabaseID:Hobt\_id (Hash value for index key) | KEY: 5:72057594044284928 (3300a4f361aa)  In this case, database ID 5 is Pubs, Hobt\_ID 72057594044284928 corresponds to non clustered index\_id 2 for object id 261575970 (**titles** table). Use the sys.partitions catalog view to associate the hobt\_id to a particular index id and object id. There is no way to unhash the index key hash to a specific index key value. |
| Row | DatabaseID:FileID:PageID:Slot(row) | RID: 5:1:104:3  In this case, database ID 5 is pubs , file ID 1 is the primary data file, page 104 is a page belonging to the titles table, and slot 3 indicates the row's position on the page. |
| Compile | DatabaseID:ObjectID [[COMPILE]] | TAB: 5:834102012 [[COMPILE]] This is not a table lock, but rather a compile lock on a stored procedure. Database ID 5 is pubs, object ID 834102012 is stored procedure usp\_myprocedure. See Knowledge Base Article 263889 for more information on blocking caused by compile locks. |

**Other columns**  
  
The remaining **sys.sysprocesses** columns can provide insight into the root of a problem as well. Their usefulness varies depending on the circumstances of the problem. For example, you can determine if the problem happens only from certain clients (hostname), on certain network libraries (net\_library), when the last batch submitted by a SPID was (last\_batch), and so on.

##### Examine the DBCC INPUTBUFFER output.

For any SPID at the head of a blocking chain or with a non-zero waittype, the blocking script will execute DBCC INPUTBUFFER to determine the current query for that SPID.  
  
In many cases, this is the query that is causing the locks that are blocking other users to be held. However, if the SPID is within a transaction, the locks may have been acquired by a previously executed query, not the current one. Therefore, you should also view the Profiler output for the SPID, not just the inputbuffer.   
  
**Note** Because the blocking script consists of multiple steps, it is possible that a SPID may appear in the first section as the head of a blocking chain, but by the time the DBCC INPUTBUFFER query is executed, it is no longer blocking and the INPUTBUFFER is not captured. This indicates that the blocking is resolving itself for that SPID and it may or may not be a problem. At this point, you can either use the fast version of the blocking script to try to ensure you capture the inputbuffer before it clears (although there is still no guarantee), or view the Profiler data from that time frame to determine what queries the SPID was executing.

#### Viewing the Profiler Data

Viewing Profiler data efficiently is extremely valuable in resolving blocking issues. The most important thing to realize is that you do not have to look at everything you captured; be selective. Profiler provides capabilities to help you effectively view the captured data. In the **Properties** dialog box (on the **File** menu, click **Properties**), Profiler allows you to limit the data displayed by removing data columns or events, grouping (sorting) by data columns and applying filters. You can search the whole trace or only a specific column for specific values (on the **Edit** menu, click **Find**). You can also save the Profiler data to a SQL Server table (on the **File** menu, point to **Save As** and then click **Table**) and run SQL queries against it.  
  
Be careful that you ***perform filtering only on a previously saved trace file***. If you perform these steps on an active trace, you risk losing data that has been captured since the trace was started. Save an active trace to a file or table first (on the **File** menu, click **Save As**) and then reopen it (on the **File** menu, click **Open**) before proceeding. When working on a saved trace file, the filtering does not permanently remove the data being filtered out, it just does not display all the data. You can add and remove events and data columns as needed to help focus your searches.  
  
**What to look for:**

* What commands has the SPID at the head of a blocking chain executed within the current transaction?  
  Filter the trace data for a particular SPID that is at the head of a blocking chain (on the **File** menu, click **Properties**; then on the **Filters** tab specify the SPID value). You can then examine the commands it has executed prior to the time it was blocking other SPIDs. If you include the Transaction events, they can easily identify when a transaction was started. Otherwise, you can search the **Text** column for BEGIN, SAVE, COMMIT, or ROLLBACK TRANSACTION operations. Use the**open\_tran** value from the **sysprocesses** table to ensure that you catch all of the transaction events. Knowing the commands executed and the transaction context will allow you to determine why a SPID is holding locks.  
    
  Remember, you can remove events and data columns. Instead of looking at both starting and completed events, choose one. If the blocking SPIDs are not stored procedures, remove the **SP:Starting** or **SP:Completed** events; the **SQLBatch** and **RPC** events will show the procedure call. Only view the SP events when you need to see that level of detail.
* What is the duration of the queries for SPIDs at the head of blocking chains?  
  If you include the completed events above, the **Duration** column will show the query execution time. This can help you identify long-running queries that are causing blocking. To determine why the query is performing slowly, view the **CPU**,**Read**, and **Writes** columns, as well as the **Execution Plan** event.

#### Categorizing Common Blocking Scenarios

The table below maps common symptoms to their probable causes. The number indicated in the **Scenario** column corresponds to the number in the "Common Blocking Scenarios and Resolutions" section of this article below. The **Waittype**, **Open\_Tran**, and **Status** columns refer to **sysprocesses** information. The **Resolves?** column indicates whether or not the blocking will resolve on its own.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scenario | Waittype | Open\_Tran | Status | Resolves? | Other Symptoms |
| 1 | Non-zero | >= 0 | runnable | Yes, when query finishes. | Physical\_IO, CPU and/or Memusage columns will increase over time. Duration for the query will be high when completed. |
| 2 | 0x0000 | >0 | sleeping | No, but SPID can be killed. | An attention signal may be seen in the Profiler trace for this SPID, indicating a query timeout or cancel has occurred. |
| 3 | 0x0000 | >= 0 | runnable | No. Will not resolve until client fetches all rows or closes connection. SPID can be killed, but it may take up to 30 seconds. | If **open\_tran** = 0, and the SPID holds locks while the transaction isolation level is default (READ COMMMITTED), this is a likely cause. |
| 4 | Varies | >= 0 | runnable | No. Will not resolve until client cancels queries or closes connections. SPIDs can be killed, but may take up to 30 seconds. | The **hostname** column in **sysprocesses** for the SPID at the head of a blocking chain will be the same as one of the SPID it is blocking. |
| 5 | 0x0000 | >0 | rollback | Yes. | An attention signal may be seen in the Profiler trace for this SPID, indicating a query timeout or cancel has occurred, or simply a rollback statement has been issued. |
| 6 | 0x0000 | >0 | sleeping | Eventually. When Windows NT determines the session is no longer active, the SQL Server connection will be broken. | The **last\_batch** value in **sysprocesses** is much earlier than the current time. |

#### Common Blocking Scenarios and Resolutions

The scenarios listed below will have the characteristics listed in the table above. This section provides additional details when applicable, as well as paths to resolution.

1. **Blocking Caused by a Normally Running Query with a Long Execution Time**  
     
   **Resolution:**  
   The solution to this type of blocking problem is to look for ways to optimize the query. Actually, this class of blocking problem may just be a performance problem, and require you to pursue it as such. For information on troubleshooting a specific slow-running query, see the following Microsoft Knowledge Base article:

[243589](http://support.microsoft.com/kb/243589) How to troubleshoot slow-running queries on SQL Server 7.0 or on later versions

For overall application performance troubleshooting, see the following Knowledge Base article:

[224587](http://support.microsoft.com/kb/224587) HOW TO: Troubleshoot Application Performance with SQL Server

For more information, see the *Performance Monitoring and Tuning How-to Topics* SQL Server 2008 Books Online topic on the following MSDN Web site:

<http://msdn.microsoft.com/en-us/library/ms187830.aspx>

If you have a long-running query that is blocking other users and cannot be optimized, consider moving it from an OLTP environment to a decision support system.

1. **Blocking Caused by a Sleeping SPID That Has Lost Track of the Transaction Nesting Level**  
     
   This type of blocking can often be identified by a SPID that is sleeping or awaiting a command, yet whose transaction nesting level (@@TRANCOUNT, **open\_tran** from **sysprocesses**) is greater than zero. This can occur if the application experiences a query timeout, or issues a cancel without also issuing the required number of ROLLBACK and/or COMMIT statements. When a SPID receives a query timeout or cancel, it will terminate the current query and batch, but does not automatically roll back or commit the transaction. The application is responsible for this, as SQL Server cannot assume that an entire transaction must be rolled back simply due to a single query being canceled. The query timeout or cancel will appear as an ATTENTION signal event for the SPID in the Profiler trace.  
     
   To demonstrate this, issue the following simple query from Query Analyzer:

BEGIN TRAN

SELECT \* FROM SYSOBJECTS S1, SYSOBJECTS S2

-- Issue this after canceling query

SELECT @@TRANCOUNT

ROLLBACK TRAN

While the query is executing, click the red **Cancel** button. After the query is canceled, SELECT @@TRANCOUNT indicates that the transaction nesting level is one. Had this been a DELETE or an UPDATE query, or had HOLDLOCK been used on the SELECT, all the locks acquired would still be held. Even with the query above, if another query had acquired and held locks earlier in the transaction, they would still be held when the above SELECT was canceled.   
  
**Resolutions:**

* + Applications must properly manage transaction nesting levels, or they may cause a blocking problem following the cancellation of the query in this manner. This can be accomplished in one of several ways:
    1. In the error handler of the client application, submit an IF @@TRANCOUNT > 0 ROLLBACK TRAN following any error, even if the client application does not believe a transaction is open. This is required, because a stored procedure called during the batch could have started a transaction without the client application's knowledge. Note that certain conditions, such as canceling the query, prevent the procedure from executing past the current statement, so even if the procedure has logic to check IF @@ERROR <> 0 and abort the transaction, this rollback code will not be executed in such cases.
    2. Use SET XACT\_ABORT ON for the connection, or in any stored procedures which begin transactions and are not cleaning up following an error. In the event of a run-time error, this setting will abort any open transactions and return control to the client. Note that T-SQL statements following the statement which caused the error will not be executed.
    3. If connection pooling is being used in an application that opens the connection and runs a small number of queries before releasing the connection back to the pool, such as a Web-based application, temporarily disabling connection pooling may help alleviate the problem until the client application is modified to handle the errors appropriately. By disabling connection pooling, releasing the connection will cause a physical logout of the SQL Server connection, resulting in the server rolling back any open transactions.
    4. If connection pooling is enabled and the destination server is SQL Server 2000, upgrading the client computer to MDAC 2.6 or later may be beneficial. This version of the MDAC components adds code to the ODBC driver and OLE DB provider so that the connection would be "reset" before it is reused. This call to sp\_reset\_connection aborts any server-initiated transactions (DTC transactions initiated by the client app are not affected), resets the default database, SET options, and so forth. Note that the connection is not reset until it is reused from the connection pool, so it is possible that a user could open a transaction and then release the connection to the connection pool, but it might not be reused for several seconds, during which time the transaction would remain open. If the connection is not reused, the transaction will be aborted when the connection times out and is removed from the connection pool. Thus, it is optimal for the client application to abort transactions in their error handler or use SET XACT\_ABORT ON to avoid this potential delay.
  + Actually, this class of blocking problem may also be a performance problem, and require you to pursue it as such. If the query execution time can be diminished, the query timeout or cancel would not occur. It is important that the application be able to handle the timeout or cancel scenarios should they arise, but you may also benefit from examining the performance of the query.

1. **Blocking Caused by a SPID Whose Corresponding Client Application Did Not Fetch All Result Rows to Completion**  
     
   After sending a query to the server, all applications must immediately fetch all result rows to completion. If an application does not fetch all result rows, locks can be left on the tables, blocking other users. If you are using an application that transparently submits SQL statements to the server, the application must fetch all result rows. If it does not (and if it cannot be configured to do so), you may be unable to resolve the blocking problem. To avoid the problem, you can restrict poorly-behaved applications to a reporting or a decision-support database.  
     
   **Resolution:**  
     
   The application must be re-written to fetch all rows of the result to completion.
2. **Blocking Caused by a Distributed Client/Server Deadlock**  
     
   Unlike a conventional deadlock, a distributed deadlock is not detectable using the RDBMS lock manager. This is due to the fact that only one of the resources involved in the deadlock is a SQL Server lock. The other side of the deadlock is at the client application level, over which SQL Server has no control. The following are two examples of how this can happen, and possible ways the application can avoid it.  
   * Client/Server Distributed Deadlock with a Single Client Thread  
     If the client has multiple open connections, and a single thread of execution, the following distributed deadlock may occur. For brevity, the term "dbproc" used here refers to the client connection structure.
   * SPID1------blocked on lock------->SPID2
   * /\ (waiting to write results
   * | back to client)
   * | |
   * | | Server side
   * | ================================|==================================
   * | <-- single thread --> | Client side
   * | \/
   * dbproc1 <------------------- dbproc2
   * (waiting to fetch (effectively blocked on dbproc1, awaiting
   * next row) single thread of execution to run)

In the case shown above, a single client application thread has two open connections. It asynchronously submits a SQL operation on dbproc1. This means it does not wait on the call to return before proceeding. The application then submits another SQL operation on dbproc2, and awaits the results to start processing the returned data. When data starts coming back (whichever dbproc first responds -- assume this is dbproc1), it processes to completion all the data returned on that dbproc. It fetches results from dbproc1 until SPID1 gets blocked on a lock held by SPID2 (because the two queries are running asynchronously on the server). At this point, dbproc1 will wait indefinitely for more data. SPID2 is not blocked on a lock, but tries to send data to its client, dbproc2. However, dbproc2 is effectively blocked on dbproc1 at the application layer as the single thread of execution for the application is in use by dbproc1. This results in a deadlock that SQL Server cannot detect or resolve because only one of the resources involved is a SQL Server resource.

* + Client/Server Distributed Deadlock with a Thread per Connection  
      
    Even if a separate thread exists for each connection on the client, a variation of this distributed deadlock may still occur as shown by the following.
  + SPID1------blocked on lock-------->SPID2
  + /\ (waiting on net write) Server side
  + | |
  + | |
  + | INSERT |SELECT
  + | ================================|==================================
  + | <-- thread per dbproc --> | Client side
  + | \/
  + dbproc1 <-----data row------- dbproc2
  + (waiting on (blocked on dbproc1, waiting for it
  + insert) to read the row from its buffer)

This case is similar to Example A, except dbproc2 and SPID2 are running a SELECT statement with the intention of performing row-at-a-time processing and handing each row through a buffer to dbproc1 for an INSERT, UPDATE, or DELETE statement on the same table. Eventually, SPID1 (performing the INSERT, UPDATE, or DELETE) becomes blocked on a lock held by SPID2 (performing the SELECT). SPID2 writes a result row to the client dbproc2. Dbproc2 then tries to pass the row in a buffer to dbproc1, but finds dbproc1 is busy (it is blocked waiting on SPID1 to finish the current INSERT, which is blocked on SPID2). At this point, dbproc2 is blocked at the application layer by dbproc1 whose SPID (SPID1) is blocked at the database level by SPID2. Again, this results in a deadlock that SQL Server cannot detect or resolve because only one of the resources involved is a SQL Server resource.

1. Both examples A and B are fundamental issues that application developers must be aware of. They must code applications to handle these cases appropriately.   
     
   **Resolutions:**  
     
   Two reliable solutions are to use either a query timeout or bound connections.  
     
   * Query Timeout  
     When a query timeout has been provided, if the distributed deadlock occurs, it will be broken when then timeout happens. See the DB-Library or ODBC documentation for more information on using a query timeout.
   * Bound Connections  
     This feature allows a client having multiple connections to bind them into a single transaction space, so the connections do not block each other. For more information, see the "Using Bound Connections" topic in SQL Server 7.0 Books Online.
2. **Blocking Caused by a SPID That Is in a "Golden," or Rollback, State**  
     
   A data modification query that is KILLed, or canceled outside of a user-defined transaction, will be rolled back. This can also occur as a side effect of the client computer restarting and its network session disconnecting. Likewise, a query selected as the deadlock victim will be rolled back. A data modification query often cannot be rolled back any faster than the changes were initially applied. For example, if a DELETE, INSERT, or UPDATE statement had been running for an hour, it could take at least an hour to roll back. This is expected behavior, because the changes made must be completely rolled back, or transactional and physical integrity in the database would be compromised. Because this must happen, SQL Server marks the SPID in a "golden" or rollback state (which means it cannot be KILLed or selected as a deadlock victim). This can often be identified by observing the output of **sp\_who**, which may indicate the ROLLBACK command. The **Status** column of **sys.sysprocesses** will indicate a ROLLBACK status, which will also appear in **sp\_who**output or in SQL Server Management Studio Activity Monitor.   
   **Resolution:**  
     
   You must wait for the SPID to finish rolling back the changes that were made.   
     
   If the server is shut down in the midst of this operation, the database will be in recovery mode upon restarting, and it will be inaccessible until all open transactions are processed. Startup recovery takes essentially the same amount of time per transaction as run-time recovery, and the database is inaccessible during this period. Thus, forcing the server down to fix a SPID in a rollback state will often be counterproductive.   
     
   To avoid this situation, do not perform large batch INSERT, UPDATE, or DELETE operations during busy hours on OLTP systems. If possible, perform such operations during periods of low activity.
3. **Blocking Caused by an Orphaned Connection**  
     
   If the client application traps or the client workstation is restarted, the network session to the server may not be immediately canceled under some conditions. From the server's perspective, the client still appears to be present, and any locks acquired may still be retained. For more information, click the following article number to view the article in the Microsoft Knowledge Base:

[137983](http://support.microsoft.com/kb/137983) How to troubleshoot orphaned connections in SQL Server

**Resolution:**  
  
If the client application has disconnected without appropriately cleaning up its resources, you can terminate the SPID by using the KILL command. The KILL command takes the SPID value as input. For example, to kill SPID 9, simply issue the following command:

KILL 9

**Note** The KILL command may take up to 30 seconds to complete, due to the interval between checks for the KILL command.

### Application Involvement in Blocking Problems

There may be a tendency to focus on server-side tuning and platform issues when facing a blocking problem. However, this does not usually lead to a resolution, and can absorb time and energy better directed at examining the client application and the queries it submits. No matter what level of visibility the application exposes regarding the database calls being made, a blocking problem nonetheless frequently requires both the inspection of the exact SQL statements submitted by the application and the application's exact behavior regarding query cancellation, connection management, fetching all result rows, and so on. If the development tool does not allow explicit control over connection management, query cancellation, query timeout, result fetching, and so on, blocking problems may not be resolvable. This potential should be closely examined before selecting an application development tool for SQL Server, especially for business-critical OLTP environments.  
  
It is vital that great care be exercised during the design and construction phase of the database and application. In particular, the resource consumption, isolation level, and transaction path length should be evaluated for each query. Each query and transaction should be as lightweight as possible. Good connection management discipline must be exercised. If this is not done, it is possible that the application may appear to have acceptable performance at low numbers of users, but the performance may degrade significantly as the number of users scales upward.  
  
With proper application and query design, Microsoft SQL Server is capable of supporting many thousands of simultaneous users on a single server, with little blocking.

# How to monitor blocking in SQL Server 2005 and in SQL Server 2000

http://support.microsoft.com/kb/271509

## [SUMMARY](javascript:void(0);)

If you are using Microsoft SQL Server 2000, you must apply SQL Server 2000 Services Pack 3a (SP3a) before you use the script that is described in this article.  
  
For more information about how to obtain SQL Server 2000 Service Pack 3a, click the following article number to view the article in the Microsoft Knowledge Base:

[290211](http://support.microsoft.com/kb/290211) How to obtain the latest SQL Server 2000 service pack

This article is an update for SQL Server 2000 or later versions of SQL Server of the following Microsoft Knowledge Base article, which applies to Microsoft SQL Server 7.0:

[251004](http://support.microsoft.com/kb/251004) How to monitor SQL Server 7.0 blocking

This article documents the usage and design of a stored procedure that you can use to diagnose blocking and performance issues. For a description of how to understand and resolve blocking, see the following article in the Microsoft Knowledge Base:

[224453](http://support.microsoft.com/kb/224453) Understanding and resolving SQL Server 7.0 or 2000 blocking problems

In Microsoft SQL Server 2005, you can also use the **Blocked Process Report** event class in SQL Server Profiler to capture the information about a task that has been blocked for more than a specified amount of time.  
  
For more information about the **Blocked Process Report** event class, visit the following Microsoft Developer Network (MSDN) Web site:

<http://msdn2.microsoft.com/en-us/library/ms191168.aspx>

## [MORE INFORMATION](javascript:void(0);)

The following description of the **sp\_blocker\_pss08** stored procedure captures this information. This information also applies to SQL Server 2005.

* The starting time (according to the computer that is running SQL Server) so that this sampling of blocking can be time-aligned with other performance information, such as a Microsoft Windows NT Performance Monitor log or a SQL Profiler log.
* Information about connections to SQL Server, by querying the **sysprocesses** system table.
* Information about lock resources, by querying the **syslockinfo** system table.
* Information about resource waits, by running DBCC SQLPERF(WAITSTATS).
* The current running SQL Server Batch for connections that are blocked by others or blocking others, by running the DBCC INPUTBUFFER statement.
* The ending time, according to the computer that is running SQL Server.

The stored procedure was created with the following optimizations to reduce the effect on performance and blocking by running this stored procedure:

* There is no output generated unless at least one connection is waiting on a resource.
* The **sysprocesses** and **syslockinfo** system tables from the **master** database are queried directly, to increase performance and to prevent this stored procedure from becoming blocked. Therefore, this stored procedure is specific to Microsoft SQL Server 2000 or later versions of SQL Server.
* A small worktable is created by the cursor to obtain the DBCC INPUTBUFFER output; this should have no noticeable effect on the usage in the **tempdb** database.
* Because the blocking can change while gathering the information, a fast mode exists, which increases performance by decreasing the results to relevant rows of the **sysprocesses** and **syslockinfo** system tables.
* If trying to track non-lock resource waits, a latch mode exists, which causes the lock output to be omitted.

This stored procedure is useful by running it from any query tool. However, Microsoft suggests that you follow these steps to perform blocking analysis:

1. Create the stored procedure **sp\_blocker\_pss08**, which is included at the end of this article, from any query tool while you are connected with a login that has **sysadmin** privileges on the SQL Server server or the instance of SQL Server you plan to monitor.
2. Create a script file with the following query to run the stored procedure in a loop. Note that the delay should be between 5 and 60   
   seconds:  
     
   WHILE 1=1  
   BEGIN  
   EXEC tempdb.dbo.sp\_blocker\_pss08   
   -- Or for fast mode   
   -- EXEC tempdb.dbo.sp\_blocker\_pss08 @fast=1  
   -- Or for latch mode   
   -- EXEC tempdb.dbo.sp\_blocker\_pss08 @latch=1  
   WAITFOR DELAY '00:00:15'  
   END  
   GO
3. This output is very useful when combined with a Microsoft Windows NT Performance Monitor log and a SQL Profiler log, so creating both at the same time is suggested. For information concerning which Profiler and Performance Monitor events to capture, and for information about how to interpret the results, see the following article in the Microsoft Knowledge Base:

[224453](http://support.microsoft.com/kb/224453) Understanding and resolving SQL Server 7.0 or 2000 blocking problems

1. Run the script file created in step 2 from either Isql.exe, the Osql.exe query tool, or the Sqlcmd utility in a Windows command prompt on the computer that is running SQL Server for which you intend to monitor to prevent networking problems from disconnecting the query tool. Following is an example command line that you can use to start Osql.exe, which assumes that the client is run from the computer that is running SQL Server, and that the script file name is Checkblk.sql. Make sure to correct the -S parameter, and replace "server" with the name of your SQL Server server (or "servername\instance" if you are monitoring a named instance) . Also correct the -i parameter, and replace "checkblk.sql" with the path and name to the script file created in step 2.

osql -E -Sserver -icheckblk.sql -ocheckblk.out -w2000

Note that you must use the other command line switches for the following reasons:

* + To prevent line wrapping in the output files which makes it easier to read.
  + To send the output to a file, specified with the -o parameter, instead of to the screen so that if the query tool has problems, you still have output until the query tool fails.

Following is the script to create the **sp\_blocker\_pss08** stored procedure:

/\*

Note: This script is meant to have 3 creations of the same stored procedure and two of them will fail

with either 207 errors or a 2714 error.

\*/

use master

GO

if exists (select \* from sysobjects where id = object\_id('dbo.sp\_blocker\_pss08') and sysstat & 0xf = 4)

drop procedure dbo.sp\_blocker\_pss08

GO

create procedure dbo.sp\_blocker\_pss08 (@latch int = 1, @fast int = 1, @appname sysname='PSSDIAG')

as

--version 19.2005 - 2005 or Later

if is\_member('sysadmin')=0

begin

print 'Must be a member of the sysadmin group in order to run this procedure'

return

end

set nocount on

SET LANGUAGE 'us\_english'

declare @spid varchar(6)

declare @blocked varchar(6)

declare @time datetime

declare @time2 datetime

declare @dbname nvarchar(128)

declare @status sql\_variant

declare @useraccess sql\_variant

declare @request varchar(12)

set @time = getdate()

declare @probclients table(spid smallint, request\_id int, ecid smallint, blocked smallint, waittype binary(2), dbid smallint,

ignore\_app tinyint, primary key (blocked, spid, request\_id, ecid))

insert @probclients select spid, request\_id, ecid, blocked, waittype, dbid,

case when convert(varchar(128),hostname) = @appname then 1 else 0 end

from master.dbo.sysprocesses where blocked!=0 or waittype != 0x0000

if exists (select spid from @probclients where ignore\_app != 1)

begin

set @time2 = getdate()

print ''

print '9.0 Start time: ' + convert(varchar(26), @time, 121) + ' ' + convert(varchar(12), datediff(ms,@time,@time2)) + ' 19.2005 '+ltrim(str(@latch))+' '+ltrim(str(@fast))

insert @probclients select distinct blocked, 0, 0, 0, 0x0000, 0, 0 from @probclients

where blocked not in (select spid from @probclients) and blocked != 0

if (@fast = 1)

begin

print ''

print 'SYSPROCESSES ' + ISNULL (@@servername,'(null)') + ' ' + str(@@microsoftversion)

select spid, status, blocked, open\_tran, waitresource, waittype,

waittime, cmd, lastwaittype, cpu, physical\_io,

memusage, last\_batch=convert(varchar(26), last\_batch,121),

login\_time=convert(varchar(26), login\_time,121),net\_address,

net\_library, dbid, ecid, kpid, hostname, hostprocess,

loginame, program\_name, nt\_domain, nt\_username, uid, sid,

sql\_handle, stmt\_start, stmt\_end, request\_id

from master.dbo.sysprocesses

where blocked!=0 or waittype != 0x0000

or spid in (select blocked from @probclients where blocked != 0)

or spid in (select spid from @probclients where blocked != 0)

print 'ESP ' + convert(varchar(12), datediff(ms,@time2,getdate()))

print ''

print 'SYSPROC FIRST PASS'

select spid, request\_id, ecid, waittype from @probclients where waittype != 0x0000

if exists(select blocked from @probclients where blocked != 0)

begin

print 'Blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

print 'SPIDs at the head of blocking chains'

select distinct spid from @probclients -- change: added distinct

where blocked = 0 and spid in (select blocked from @probclients where spid != 0)

if @latch = 0 and exists (select spid from @probclients where waittype between 0x0001 and 0x0017) -- Change: exists

begin

print 'SYSLOCKINFO'

select @time2 = getdate()

select spid = convert (smallint, req\_spid),

ecid = convert (smallint, req\_ecid),

rsc\_dbid As dbid,

rsc\_objid As ObjId,

rsc\_indid As IndId,

Type = case rsc\_type when 1 then 'NUL'

when 2 then 'DB'

when 3 then 'FIL'

when 4 then 'IDX'

when 5 then 'TAB'

when 6 then 'PAG'

when 7 then 'KEY'

when 8 then 'EXT'

when 9 then 'RID'

when 10 then 'APP' end,

Resource = substring (rsc\_text, 1, 16),

Mode = case req\_mode + 1 when 1 then NULL

when 2 then 'Sch-S'

when 3 then 'Sch-M'

when 4 then 'S'

when 5 then 'U'

when 6 then 'X'

when 7 then 'IS'

when 8 then 'IU'

when 9 then 'IX'

when 10 then 'SIU'

when 11 then 'SIX'

when 12 then 'UIX'

when 13 then 'BU'

when 14 then 'RangeS-S'

when 15 then 'RangeS-U'

when 16 then 'RangeIn-Null'

when 17 then 'RangeIn-S'

when 18 then 'RangeIn-U'

when 19 then 'RangeIn-X'

when 20 then 'RangeX-S'

when 21 then 'RangeX-U'

when 22 then 'RangeX-X'end,

Status = case req\_status when 1 then 'GRANT'

when 2 then 'CNVT'

when 3 then 'WAIT' end,

req\_transactionID As TransID, req\_transactionUOW As TransUOW

from master.dbo.syslockinfo s,

@probclients p

where p.spid = s.req\_spid

--and ((p.waittype between 0x0001 and 0x0017) or ()) --change: added line

print 'ESL ' + convert(varchar(12), datediff(ms,@time2,getdate()))

end -- latch not set

end

else

print 'No blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

end -- fast set

else

begin -- Fast not set

print ''

print 'SYSPROCESSES ' + ISNULL (@@servername,'(null)') + ' ' + str(@@microsoftversion)

select spid, status, blocked, open\_tran, waitresource, waittype,

waittime, cmd, lastwaittype, cpu, physical\_io,

memusage, last\_batch=convert(varchar(26), last\_batch,121),

login\_time=convert(varchar(26), login\_time,121),net\_address,

net\_library, dbid, ecid, kpid, hostname, hostprocess,

loginame, program\_name, nt\_domain, nt\_username, uid, sid,

sql\_handle, stmt\_start, stmt\_end, request\_id

from master.dbo.sysprocesses

print 'ESP ' + convert(varchar(12), datediff(ms,@time2,getdate()))

print ''

print 'SYSPROC FIRST PASS'

select spid, request\_id, ecid, waittype from @probclients where waittype != 0x0000

if exists(select blocked from @probclients where blocked != 0)

begin

print 'Blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

print 'SPIDs at the head of blocking chains'

select spid from @probclients

where blocked = 0 and spid in (select blocked from @probclients where spid != 0)

if @latch = 0

begin

print 'SYSLOCKINFO'

select @time2 = getdate()

select spid = convert (smallint, req\_spid),

ecid = convert (smallint, req\_ecid),

rsc\_dbid As dbid,

rsc\_objid As ObjId,

rsc\_indid As IndId,

Type = case rsc\_type when 1 then 'NUL'

when 2 then 'DB'

when 3 then 'FIL'

when 4 then 'IDX'

when 5 then 'TAB'

when 6 then 'PAG'

when 7 then 'KEY'

when 8 then 'EXT'

when 9 then 'RID'

when 10 then 'APP' end,

Resource = substring (rsc\_text, 1, 16),

Mode = case req\_mode + 1 when 1 then NULL

when 2 then 'Sch-S'

when 3 then 'Sch-M'

when 4 then 'S'

when 5 then 'U'

when 6 then 'X'

when 7 then 'IS'

when 8 then 'IU'

when 9 then 'IX'

when 10 then 'SIU'

when 11 then 'SIX'

when 12 then 'UIX'

when 13 then 'BU'

when 14 then 'RangeS-S'

when 15 then 'RangeS-U'

when 16 then 'RangeIn-Null'

when 17 then 'RangeIn-S'

when 18 then 'RangeIn-U'

when 19 then 'RangeIn-X'

when 20 then 'RangeX-S'

when 21 then 'RangeX-U'

when 22 then 'RangeX-X'end,

Status = case req\_status when 1 then 'GRANT'

when 2 then 'CNVT'

when 3 then 'WAIT' end,

req\_transactionID As TransID, req\_transactionUOW As TransUOW

from master.dbo.syslockinfo

print 'ESL ' + convert(varchar(12), datediff(ms,@time2,getdate()))

end -- latch not set

end

else

print 'No blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

end -- Fast not set

print 'sys.dm\_os\_wait\_stats'

select \* from sys.dm\_os\_wait\_stats where waiting\_tasks\_count > 0

print 'OWS'

Print ''

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

Print 'Print out DBCC Input buffer for all blocked or blocking spids.'

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

declare ibuffer cursor fast\_forward for

select distinct cast (spid as varchar(6)) as spid, cast (request\_id as varchar(12)) as request\_id

from @probclients

where (spid <> @@spid) and (spid > 50) and

((blocked!=0 or (waittype != 0x0000 and ignore\_app = 0))

or spid in (select blocked from @probclients where blocked != 0))

open ibuffer

fetch next from ibuffer into @spid, @request

while (@@fetch\_status != -1)

begin

print ''

print 'DBCC INPUTBUFFER FOR SPID ' + @spid +'('+@request+')'

exec ('dbcc inputbuffer (' + @spid + ',' + @request +')')

fetch next from ibuffer into @spid, @request

end

deallocate ibuffer

Print ''

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

Print 'Print out DBCC OPENTRAN for active databases for all blocked or blocking spids.'

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

declare ibuffer cursor fast\_forward for

select distinct cast (dbid as varchar(6)) from @probclients

where dbid != 0

open ibuffer

fetch next from ibuffer into @spid

while (@@fetch\_status != -1)

begin

print ''

set @dbname = db\_name(@spid)

set @status = DATABASEPROPERTYEX(@dbname,'Status')

set @useraccess = DATABASEPROPERTYEX(@dbname,'UserAccess')

print 'DBCC OPENTRAN FOR DBID ' + @spid + ' ['+ @dbname + ']'

if @status = N'ONLINE' and @useraccess != N'SINGLE\_USER'

dbcc opentran(@dbname)

else

print 'Skipped: Status=' + convert(nvarchar(128),@status)

+ ' UserAccess=' + convert(nvarchar(128),@useraccess)

print ''

if @spid = '2' select @blocked = 'Y'

fetch next from ibuffer into @spid

end

deallocate ibuffer

if @blocked != 'Y'

begin

print ''

print 'DBCC OPENTRAN FOR DBID 2 [tempdb]'

dbcc opentran ('tempdb')

end

print 'End time: ' + convert(varchar(26), getdate(), 121)

end -- All

else

print '8 No Waittypes: ' + convert(varchar(26), @time, 121) + ' '

+ convert(varchar(12), datediff(ms,@time,getdate())) + ' ' + ISNULL (@@servername,'(null)') + ' 19.2005'

GO

create procedure dbo.sp\_blocker\_pss08 (@latch int = 1, @fast int = 1, @appname sysname='PSSDIAG')

as

--version 19.20003 --2000 SP3 or Later

if is\_member('sysadmin')=0

begin

print 'Must be a member of the sysadmin group in order to run this procedure'

return

end

set nocount on

SET LANGUAGE 'us\_english'

declare @spid varchar(6)

declare @blocked varchar(6)

declare @time datetime

declare @time2 datetime

declare @dbname nvarchar(128)

declare @status sql\_variant

declare @useraccess sql\_variant

set @time = getdate()

declare @probclients table(spid smallint, ecid smallint, blocked smallint, waittype binary(2), dbid smallint,

ignore\_app tinyint, primary key (blocked, spid, ecid))

insert @probclients select spid, ecid, blocked, waittype, dbid,

case when convert(varchar(128),hostname) = @appname then 1 else 0 end

from master.dbo.sysprocesses where blocked!=0 or waittype != 0x0000

if exists (select spid from @probclients where ignore\_app != 1 or waittype != 0x020B)

begin

set @time2 = getdate()

print ''

print '8.2 Start time: ' + convert(varchar(26), @time, 121) + ' ' + convert(varchar(12), datediff(ms,@time,@time2)) + ' 19.20003 ' +ltrim(str(@latch))+' '+ltrim(str(@fast))

insert @probclients select distinct blocked, 0, 0, 0x0000, 0, 0 from @probclients

where blocked not in (select spid from @probclients) and blocked != 0

if (@fast = 1)

begin

print ''

print 'SYSPROCESSES ' + ISNULL (@@servername,'(null)') + ' ' + str(@@microsoftversion)

select spid, status, blocked, open\_tran, waitresource, waittype,

waittime, cmd, lastwaittype, cpu, physical\_io,

memusage, last\_batch=convert(varchar(26), last\_batch,121),

login\_time=convert(varchar(26), login\_time,121),net\_address,

net\_library, dbid, ecid, kpid, hostname, hostprocess,

loginame, program\_name, nt\_domain, nt\_username, uid, sid,

sql\_handle, stmt\_start, stmt\_end

from master.dbo.sysprocesses

where blocked!=0 or waittype != 0x0000

or spid in (select blocked from @probclients where blocked != 0)

or spid in (select spid from @probclients where blocked != 0)

print 'ESP ' + convert(varchar(12), datediff(ms,@time2,getdate()))

print ''

print 'SYSPROC FIRST PASS'

select spid, ecid, waittype from @probclients where waittype != 0x0000

if exists(select blocked from @probclients where blocked != 0)

begin

print 'Blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

print 'SPIDs at the head of blocking chains'

select spid from @probclients

where blocked = 0 and spid in (select blocked from @probclients where spid != 0)

if @latch = 0

begin

print 'SYSLOCKINFO'

select @time2 = getdate()

select spid = convert (smallint, req\_spid),

ecid = convert (smallint, req\_ecid),

rsc\_dbid As dbid,

rsc\_objid As ObjId,

rsc\_indid As IndId,

Type = case rsc\_type when 1 then 'NUL'

when 2 then 'DB'

when 3 then 'FIL'

when 4 then 'IDX'

when 5 then 'TAB'

when 6 then 'PAG'

when 7 then 'KEY'

when 8 then 'EXT'

when 9 then 'RID'

when 10 then 'APP' end,

Resource = substring (rsc\_text, 1, 16),

Mode = case req\_mode + 1 when 1 then NULL

when 2 then 'Sch-S'

when 3 then 'Sch-M'

when 4 then 'S'

when 5 then 'U'

when 6 then 'X'

when 7 then 'IS'

when 8 then 'IU'

when 9 then 'IX'

when 10 then 'SIU'

when 11 then 'SIX'

when 12 then 'UIX'

when 13 then 'BU'

when 14 then 'RangeS-S'

when 15 then 'RangeS-U'

when 16 then 'RangeIn-Null'

when 17 then 'RangeIn-S'

when 18 then 'RangeIn-U'

when 19 then 'RangeIn-X'

when 20 then 'RangeX-S'

when 21 then 'RangeX-U'

when 22 then 'RangeX-X'end,

Status = case req\_status when 1 then 'GRANT'

when 2 then 'CNVT'

when 3 then 'WAIT' end,

req\_transactionID As TransID, req\_transactionUOW As TransUOW

from master.dbo.syslockinfo s,

@probclients p

where p.spid = s.req\_spid

print 'ESL ' + convert(varchar(12), datediff(ms,@time2,getdate()))

end -- latch not set

end

else

print 'No blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

end -- fast set

else

begin -- Fast not set

print ''

print 'SYSPROCESSES ' + ISNULL (@@servername,'(null)') + ' ' + str(@@microsoftversion)

select spid, status, blocked, open\_tran, waitresource, waittype,

waittime, cmd, lastwaittype, cpu, physical\_io,

memusage, last\_batch=convert(varchar(26), last\_batch,121),

login\_time=convert(varchar(26), login\_time,121),net\_address,

net\_library, dbid, ecid, kpid, hostname, hostprocess,

loginame, program\_name, nt\_domain, nt\_username, uid, sid,

sql\_handle, stmt\_start, stmt\_end

from master.dbo.sysprocesses

print 'ESP ' + convert(varchar(12), datediff(ms,@time2,getdate()))

print ''

print 'SYSPROC FIRST PASS'

select spid, ecid, waittype from @probclients where waittype != 0x0000

if exists(select blocked from @probclients where blocked != 0)

begin

print 'Blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

print 'SPIDs at the head of blocking chains'

select spid from @probclients

where blocked = 0 and spid in (select blocked from @probclients where spid != 0)

if @latch = 0

begin

print 'SYSLOCKINFO'

select @time2 = getdate()

select spid = convert (smallint, req\_spid),

ecid = convert (smallint, req\_ecid),

rsc\_dbid As dbid,

rsc\_objid As ObjId,

rsc\_indid As IndId,

Type = case rsc\_type when 1 then 'NUL'

when 2 then 'DB'

when 3 then 'FIL'

when 4 then 'IDX'

when 5 then 'TAB'

when 6 then 'PAG'

when 7 then 'KEY'

when 8 then 'EXT'

when 9 then 'RID'

when 10 then 'APP' end,

Resource = substring (rsc\_text, 1, 16),

Mode = case req\_mode + 1 when 1 then NULL

when 2 then 'Sch-S'

when 3 then 'Sch-M'

when 4 then 'S'

when 5 then 'U'

when 6 then 'X'

when 7 then 'IS'

when 8 then 'IU'

when 9 then 'IX'

when 10 then 'SIU'

when 11 then 'SIX'

when 12 then 'UIX'

when 13 then 'BU'

when 14 then 'RangeS-S'

when 15 then 'RangeS-U'

when 16 then 'RangeIn-Null'

when 17 then 'RangeIn-S'

when 18 then 'RangeIn-U'

when 19 then 'RangeIn-X'

when 20 then 'RangeX-S'

when 21 then 'RangeX-U'

when 22 then 'RangeX-X'end,

Status = case req\_status when 1 then 'GRANT'

when 2 then 'CNVT'

when 3 then 'WAIT' end,

req\_transactionID As TransID, req\_transactionUOW As TransUOW

from master.dbo.syslockinfo

print 'ESL ' + convert(varchar(12), datediff(ms,@time2,getdate()))

end -- latch not set

end

else

print 'No blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

end -- Fast not set

print 'DBCC SQLPERF(WAITSTATS)'

dbcc sqlperf(waitstats)

Print ''

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

Print 'Print out DBCC Input buffer for all blocked or blocking spids.'

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

declare ibuffer cursor fast\_forward for

select distinct cast (spid as varchar(6)) as spid

from @probclients

where (spid <> @@spid) and

((blocked!=0 or (waittype != 0x0000 and ignore\_app = 0))

or spid in (select blocked from @probclients where blocked != 0))

open ibuffer

fetch next from ibuffer into @spid

while (@@fetch\_status != -1)

begin

print ''

print 'DBCC INPUTBUFFER FOR SPID ' + @spid

exec ('dbcc inputbuffer (' + @spid + ')')

fetch next from ibuffer into @spid

end

deallocate ibuffer

Print ''

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

Print 'Print out DBCC OPENTRAN for active databases for all blocked or blocking spids.'

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

declare ibuffer cursor fast\_forward for

select distinct cast (dbid as varchar(6)) from @probclients

where dbid != 0

open ibuffer

fetch next from ibuffer into @spid

while (@@fetch\_status != -1)

begin

print ''

set @dbname = db\_name(@spid)

set @status = DATABASEPROPERTYEX(@dbname,'Status')

set @useraccess = DATABASEPROPERTYEX(@dbname,'UserAccess')

print 'DBCC OPENTRAN FOR DBID ' + @spid + ' ['+ @dbname + ']'

if @status = N'ONLINE' and @useraccess != N'SINGLE\_USER'

dbcc opentran(@dbname)

else

print 'Skipped: Status=' + convert(nvarchar(128),@status)

+ ' UserAccess=' + convert(nvarchar(128),@useraccess)

print ''

if @spid = '2' select @blocked = 'Y'

fetch next from ibuffer into @spid

end

deallocate ibuffer

if @blocked != 'Y'

begin

print ''

print 'DBCC OPENTRAN FOR DBID 2 [tempdb]'

dbcc opentran ('tempdb')

end

print 'End time: ' + convert(varchar(26), getdate(), 121)

end -- All

else

print '8 No Waittypes: ' + convert(varchar(26), @time, 121) + ' '

+ convert(varchar(12), datediff(ms,@time,getdate())) + ' ' + ISNULL (@@servername,'(null)') + ' 19.20003'

GO

create procedure dbo.sp\_blocker\_pss08 (@latch int = 1, @fast int = 1, @appname sysname='PSSDIAG')

as

--version 19.2000 -- 2000 before SP3

if is\_member('sysadmin')=0

begin

print 'Must be a member of the sysadmin group in order to run this procedure'

return

end

set nocount on

declare @spid varchar(6)

declare @blocked varchar(6)

declare @time datetime

declare @time2 datetime

declare @dbname nvarchar(128)

declare @status sql\_variant

declare @useraccess sql\_variant

set @time = getdate()

declare @probclients table(spid smallint, ecid smallint, blocked smallint, waittype binary(2), dbid smallint,

ignore\_app tinyint, primary key (blocked, spid, ecid))

insert @probclients select spid, ecid, blocked, waittype, dbid,

case when convert(varchar(128),hostname) = @appname then 1 else 0 end

from master.dbo.sysprocesses where blocked!=0 or waittype != 0x0000

if exists (select spid from @probclients where ignore\_app != 1 or waittype != 0x020B)

begin

set @time2 = getdate()

print ''

print '8 Start time: ' + convert(varchar(26), @time, 121) + ' ' + convert(varchar(12), datediff(ms,@time,@time2)) + ' 19.2000 '+ltrim(str(@latch))+' '+ltrim(str(@fast))

insert @probclients select distinct blocked, 0, 0, 0x0000, 0, 0 from @probclients

where blocked not in (select spid from @probclients) and blocked != 0

if (@fast = 1)

begin

print ''

print 'SYSPROCESSES ' + ISNULL (@@servername,'(null)') + ' ' + str(@@microsoftversion)

select spid, status, blocked, open\_tran, waitresource, waittype,

waittime, cmd, lastwaittype, cpu, physical\_io,

memusage,last\_batch=convert(varchar(26), last\_batch,121),

login\_time=convert(varchar(26), login\_time,121), net\_address,

net\_library, dbid, ecid, kpid, hostname, hostprocess,

loginame, program\_name, nt\_domain, nt\_username, uid, sid

from master.dbo.sysprocesses

where blocked!=0 or waittype != 0x0000

or spid in (select blocked from @probclients where blocked != 0)

or spid in (select spid from @probclients where waittype != 0x0000)

print 'ESP ' + convert(varchar(12), datediff(ms,@time2,getdate()))

print ''

print 'SYSPROC FIRST PASS'

select spid, ecid, waittype from @probclients where waittype != 0x0000

if exists(select blocked from @probclients where blocked != 0)

begin

print 'Blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

print 'SPIDs at the head of blocking chains'

select spid from @probclients

where blocked = 0 and spid in (select blocked from @probclients where spid != 0)

if @latch = 0

begin

print 'SYSLOCKINFO'

select @time2 = getdate()

select spid = convert (smallint, req\_spid),

ecid = convert (smallint, req\_ecid),

rsc\_dbid As dbid,

rsc\_objid As ObjId,

rsc\_indid As IndId,

Type = case rsc\_type when 1 then 'NUL'

when 2 then 'DB'

when 3 then 'FIL'

when 4 then 'IDX'

when 5 then 'TAB'

when 6 then 'PAG'

when 7 then 'KEY'

when 8 then 'EXT'

when 9 then 'RID'

when 10 then 'APP' end,

Resource = substring (rsc\_text, 1, 16),

Mode = case req\_mode + 1 when 1 then NULL

when 2 then 'Sch-S'

when 3 then 'Sch-M'

when 4 then 'S'

when 5 then 'U'

when 6 then 'X'

when 7 then 'IS'

when 8 then 'IU'

when 9 then 'IX'

when 10 then 'SIU'

when 11 then 'SIX'

when 12 then 'UIX'

when 13 then 'BU'

when 14 then 'RangeS-S'

when 15 then 'RangeS-U'

when 16 then 'RangeIn-Null'

when 17 then 'RangeIn-S'

when 18 then 'RangeIn-U'

when 19 then 'RangeIn-X'

when 20 then 'RangeX-S'

when 21 then 'RangeX-U'

when 22 then 'RangeX-X'end,

Status = case req\_status when 1 then 'GRANT'

when 2 then 'CNVT'

when 3 then 'WAIT' end,

req\_transactionID As TransID, req\_transactionUOW As TransUOW

from master.dbo.syslockinfo s,

@probclients p

where p.spid = s.req\_spid

print 'ESL ' + convert(varchar(12), datediff(ms,@time2,getdate()))

end -- latch not set

end

else

print 'No blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

end -- fast set

else

begin -- Fast not set

print ''

print 'SYSPROCESSES ' + ISNULL (@@servername,'(null)') + ' ' + str(@@microsoftversion)

select spid, status, blocked, open\_tran, waitresource, waittype,

waittime, cmd, lastwaittype, cpu, physical\_io,

memusage,last\_batch=convert(varchar(26), last\_batch,121),

login\_time=convert(varchar(26), login\_time,121), net\_address,

net\_library, dbid, ecid, kpid, hostname, hostprocess,

loginame, program\_name, nt\_domain, nt\_username, uid, sid

from master.dbo.sysprocesses

print 'ESP ' + convert(varchar(12), datediff(ms,@time2,getdate()))

print ''

print 'SYSPROC FIRST PASS'

select spid, ecid, waittype from @probclients where waittype != 0x0000

if exists(select blocked from @probclients where blocked != 0)

begin

print 'Blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

print 'SPIDs at the head of blocking chains'

select spid from @probclients

where blocked = 0 and spid in (select blocked from @probclients where spid != 0)

if @latch = 0

begin

print 'SYSLOCKINFO'

select @time2 = getdate()

select spid = convert (smallint, req\_spid),

ecid = convert (smallint, req\_ecid),

rsc\_dbid As dbid,

rsc\_objid As ObjId,

rsc\_indid As IndId,

Type = case rsc\_type when 1 then 'NUL'

when 2 then 'DB'

when 3 then 'FIL'

when 4 then 'IDX'

when 5 then 'TAB'

when 6 then 'PAG'

when 7 then 'KEY'

when 8 then 'EXT'

when 9 then 'RID'

when 10 then 'APP' end,

Resource = substring (rsc\_text, 1, 16),

Mode = case req\_mode + 1 when 1 then NULL

when 2 then 'Sch-S'

when 3 then 'Sch-M'

when 4 then 'S'

when 5 then 'U'

when 6 then 'X'

when 7 then 'IS'

when 8 then 'IU'

when 9 then 'IX'

when 10 then 'SIU'

when 11 then 'SIX'

when 12 then 'UIX'

when 13 then 'BU'

when 14 then 'RangeS-S'

when 15 then 'RangeS-U'

when 16 then 'RangeIn-Null'

when 17 then 'RangeIn-S'

when 18 then 'RangeIn-U'

when 19 then 'RangeIn-X'

when 20 then 'RangeX-S'

when 21 then 'RangeX-U'

when 22 then 'RangeX-X'end,

Status = case req\_status when 1 then 'GRANT'

when 2 then 'CNVT'

when 3 then 'WAIT' end,

req\_transactionID As TransID, req\_transactionUOW As TransUOW

from master.dbo.syslockinfo

print 'ESL ' + convert(varchar(12), datediff(ms,@time2,getdate()))

end -- latch not set

end

else

print 'No blocking via locks at ' + convert(varchar(26), @time, 121)

print ''

end -- Fast not set

print 'DBCC SQLPERF(WAITSTATS)'

dbcc sqlperf(waitstats)

Print ''

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

Print 'Print out DBCC Input buffer for all blocked or blocking spids.'

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

declare ibuffer cursor fast\_forward for

select distinct cast (spid as varchar(6)) as spid

from @probclients

where (spid <> @@spid) and

((blocked!=0 or (waittype != 0x0000 and ignore\_app = 0))

or spid in (select blocked from @probclients where blocked != 0))

open ibuffer

fetch next from ibuffer into @spid

while (@@fetch\_status != -1)

begin

print ''

print 'DBCC INPUTBUFFER FOR SPID ' + @spid

exec ('dbcc inputbuffer (' + @spid + ')')

fetch next from ibuffer into @spid

end

deallocate ibuffer

Print ''

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

Print 'Print out DBCC OPENTRAN for active databases for all blocked or blocking spids.'

Print '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

declare ibuffer cursor fast\_forward for

select distinct cast (dbid as varchar(6)) from @probclients

where dbid != 0

open ibuffer

fetch next from ibuffer into @spid

while (@@fetch\_status != -1)

begin

print ''

set @dbname = db\_name(@spid)

set @status = DATABASEPROPERTYEX(@dbname,'Status')

set @useraccess = DATABASEPROPERTYEX(@dbname,'UserAccess')

print 'DBCC OPENTRAN FOR DBID ' + @spid + ' ['+ @dbname + ']'

if @status = N'ONLINE' and @useraccess != N'SINGLE\_USER'

dbcc opentran(@dbname)

else

print 'Skipped: Status=' + convert(nvarchar(128),@status)

+ ' UserAccess=' + convert(nvarchar(128),@useraccess)

print ''

if @spid = '2' select @blocked = 'Y'

fetch next from ibuffer into @spid

end

deallocate ibuffer

if @blocked != 'Y'

begin

print ''

print 'DBCC OPENTRAN FOR DBID 2 [tempdb]'

dbcc opentran ('tempdb')

end

print 'End time: ' + convert(varchar(26), getdate(), 121)

end -- All

else

print '8 No Waittypes: ' + convert(varchar(26), @time, 121) + ' '

+ convert(varchar(12), datediff(ms,@time,getdate())) + ' ' + ISNULL (@@servername,'(null)') + ' 19.2000'

GO

For more information, visit the following Web site:

<http://blogs.msdn.com/psssql/archive/2007/02/21/sql-server-2005-performance-statistics-script.aspx>

# How to troubleshoot slow-running queries on SQL Server 7.0 or on later versions

http://support.microsoft.com/kb/243589

## [Summary](javascript:void(0);)

This article describes how to handle a performance issue that applications may experience in conjunction with Microsoft SQL Server: slow performance of a specific query or group of queries. If you are troubleshooting a performance issue, but you have not isolated the problem to a specific query or small group of queries that perform slower than expected, see the following article in the Microsoft Knowledge Base before you continue:

[224587](http://support.microsoft.com/kb/224587) HOW TO: Troubleshoot Application Performance with SQL Server

This article is based on the assumption that you have used the article 224587 to narrow down the scope of the problem, and that you have captured a SQL Profiler trace with the specific events and data columns that are detailed in the article 224587.  
  
Tuning database queries can be a multi-faceted endeavor. The following sections discuss common items to examine when you are investigating query performance.  
  
**Note** If you are using SQL Server 2005, use SQL Server Management Studio instead of SQL Query Analyzer, and use Database Engine Tuning Advisor instead of the Index Tuning Wizard.

### Verify the Existence of the Correct Indexes

One of the first checks to perform when you are experiencing slow query execution times is an index analysis. If you are investigating a single query, you can use the **Perform Index Analysis** option in SQL Query Analyzer; if you have a SQL Profiler trace of a large workload, you can use the Index Tuning Wizard. Both methods use the SQL Server query optimizer to determine which indexes would be helpful for the specified queries. This is a very efficient method for determining whether the correct indexes exist in your database.   
  
For information about how to use the Index Tuning Wizard, see the "Index Tuning Wizard" topic in SQL Server 7.0 Books Online.  
  
If you have upgraded your application from a previous version of SQL Server, different indexes may be more efficient in SQL Server 7.0 because of optimizer and storage engine changes. The Index Tuning Wizard helps you to determine if a change in indexing strategy would improve performance.   
  
For more information about how to use Database Engine Tuning Advisor instead of the Index Tuning Wizard in SQL Server 2005, see the following topics in SQL Server 2005 Books Online:

* Differences between Database Engine Tuning Advisor and Index Tuning Wizard
* Database Engine Tuning Advisor tutorial

### Remove All Query, Table, and Join Hints

Hints override query optimization and can prevent the query optimizer from choosing the fastest execution plan. Because of optimizer changes, hints that improved performance in earlier versions of SQL Server may have no effect or may actually adversely affect performance in SQL Server 7.0. Additionally, join hints can cause performance degradation based on the following reasons:

* Join hints prevent an ad hoc query from being eligible for auto-parameterization and caching of the query plan.
* When you use a join hint, it implies that you want to force the join order for all tables in the query, even if those joins do not explicitly use a hint.

If the query that you are analyzing includes any hints, remove them, and then re-evaluate the performance. 

### Examine the Execution Plan

After you confirm that the correct indexes exist, and that no hints are restricting the optimizer's ability to generate an efficient plan, you can examine the query execution plan. You can use any of the following methods to view the execution plan for a query:

* **SQL Profiler**  
    
  If you captured the **MISC:Execution Plan** event in SQL Profiler, it will occur immediately before the **StmtCompleted** event for the query for the particular system process ID (SPID).
* **SQL Query Analyzer: Graphical Showplan**  
    
  With the query selected in the query window, click the **Query** menu, and then click **Display Estimated Execution Plan**.   
    
  **NOTE**: If the stored procedure or batch creates and references temporary tables, you must use a SET STATISTICS PROFILE ON statement or explicitly create the temporary tables before you display the execution plan.
* **SHOWPLAN\_ALL and SHOWPLAN\_TEXT**  
    
  To receive a text version of the estimated execution plan, you can use the SET SHOWPLAN\_ALL and SET SHOWPLAN\_TEXT options. See the "SET SHOWPLAN\_ALL (T-SQL)" and "SET SHOWPLAN\_TEXT (T-SQL)" topics in SQL Server 7.0 Books Online for more details.  
    
  **NOTE**: If the stored procedure or batch creates and references temporary tables, you must use the SET STATISTICS PROFILE ON option or explicitly create the temporary tables before displaying the execution plan.
* **STATISTICS PROFILE**  
    
  When you are displaying the estimated execution plan, either graphically or by using SHOWPLAN, the query is not actually executed. Therefore, if you create temporary tables in a batch or a stored procedure, you cannot display the estimated execution plans because the temporary tables will not exist. STATISTICS PROFILE executes the query first, and then displays the actual execution plan. See the "SET STATISTICS PROFILE (T-SQL)" topic in SQL Server 7.0 Books Online for more details. When it is running in SQL Query Analyzer, this appears in graphical format on the **Execution Plan** tab in the results pane.

For more information about how to display the estimated execution plan in SQL Server 2005, see the "How to display the estimated execution plan" topic in SQL Server 2005 Books Online.

#### Examine the Showplan Output

Showplan output provides a lot of information about the execution plan that SQL Server is using for a particular query. The details of the information and events that are generated are discussed in detail in the "Optimizing Database Performance" chapter of SQL Server 7.0 Books Online. The following are some basic aspects of the execution plan that you can view to determine whether you are using the best plan:

* Correct Index Usage  
    
  The showplan output displays each table that is involved in the query and the access path that is used to obtain data from it. With graphical showplan, move the pointer over a table to see the details for each table. If an index is in use, you see "Index Seek"; if an index is not in use, you see either "Table Scan" for a heap or "Clustered Index Scan" for a table that has a clustered index. "Clustered Index Scan" indicates that the table is being scanned through the clustered index, *not* that the clustered index is being used to directly access individual rows.   
    
  If you determine that a useful index exists and it is not being used for the query, you can try forcing the index by using an index hint. See the "FROM (T-SQL)" topic in SQL Server Books Online for more details about index hints.
* Correct Join Order  
    
  The showplan output indicates in what order tables that are involved in a query are being joined. For nested loop joins, the upper table that is listed is the outer table and it should be the smaller of the two tables. For hash joins, the upper table becomes the build input and should also be the smaller of the two tables. However, note that the order is less critical because the query processor can reverse build and probe inputs at run time if it finds that the optimizer made a wrong decision. You can determine which table returns fewer rows by checking the Row Count estimates in the showplan output.  
    
  If you determine that the query may benefit from a different join order, you can try forcing the join order with a join hint. See the "FROM (T-SQL)" topic in SQL Server 7.0 Books Online for more details about join hints.   
    
  **NOTE**: Using a join hint in a large query implicitly forces the join order for the other tables in the query as if FORCEPLAN was set.
* Correct Join Type  
    
  SQL Server uses nested loop, hash, and merge joins. If a slow-performing query is using one join technique over another, you can try forcing a different join type. For example, if a query is using a hash join, you can force a nested loops join by using the LOOP join hint. See the "FROM (T-SQL)" topic in SQL Server 7.0 Books Online for more details on join hints.   
    
  **NOTE**: Using a join hint in a large query implicitly forces the join type for the other tables in the query as if FORCEPLAN was set.
* Parallel Execution  
    
  If you are using a multiprocessor computer, you can also investigate whether a parallel plan is in use. If parallelism is in use, you see a **PARALLELISM** (Gather Streams) event. If a particular query is slow when it is using a parallel plan, you can try forcing a non-parallel plan by using the OPTION (MAXDOP 1) hint. See the "SELECT (T-SQL)" topic in SQL Server 7.0 Books Online for more details.

For more information about how to use Showplan execution plan output in SQL Server 2005, see the following topics in SQL Server 2005 Books Online:

* How to save an execution plan in XML format
* XML Showplans
* Showplan security

**CAUTION**: Because the query optimizer typically selects the best execution plan for a query, Microsoft recommends that you use join hints, query hints, and table hints only as a last resort, and only if you are an experienced database administrators.

# How to: Troubleshoot Application Performance with SQL Server

http://support.microsoft.com/kb/224587

## [SUMMARY](javascript:void(0);)

This step-by-step article describes how to troubleshoot SQL Server performance issues. Troubleshooting performance issues involves the use of a series of steps to isolate and determine the cause of an application slowdown. Possible causes include:

* Blocking.
* System resource contention.
* Application design problems.
* A particular set of queries or stored procedures with long execution times.

This article describes how to determine the source of a performance issue. It also references other articles in the Microsoft Knowledge Base that cover the details of specific performance issues for additional troubleshooting.

### SQL Profiler

SQL Profiler is a powerful tool when troubleshooting your SQL Server 7.0, or later, application performance. SQL Profiler allows you to easily capture all the events that are occurring on the server under a typical load, and it provides information about them. Using SQL Profiler in conjunction with Microsoft Windows NT Performance Monitor and some simple queries to identify whether blocking is occurring will give you the information you must have to resolve the vast majority of performance problems.

### What to Monitor

1. Set up SQL Profiler to capture a trace. To do so, follow these steps:

1. Open SQL Profiler.
2. On the **Tools** menu, click **Options**.
3. Make sure that the **All Event Classes** and the **All Data Columns** options are selected.
4. Click **OK**.
5. Create a new trace.
6. On the **File** menu, point to **New**, and then click **Trace**.
7. On the **General** tab, specify a trace name and a file to capture the data to.
8. On the **Events** tab, add the following event types to your trace:

|  |  |  |
| --- | --- | --- |
| Heading | Event to add | Description |
| Cursors | CursorPrepare | This event indicates that a cursor on an SQL statement has been prepared by using ODBC, OLEDB, or the DB-Library. |
| Error and Warning | Missing Column Statistics | This event indicates column statistics that might have been useful for the Optimizer were not available. The **Text** column shows the list of columns with missing statistics. This event, in conjunction with a **Misc: Auto-UpdateStats** event, indicates that the **Auto Create Statistics** option was triggered. |
| Misc. | Attention | This event indicates that an attention signal was sent by a client. |
| Misc. | Auto-UpdateStats | This event indicates that the **Auto Update Statistics** option was triggered. |
| Misc. | Exec Prepared SQL | This event indicates that ODBC, OLE DB, or the DB-Library executed a previously prepared Transact-SQL statement, or statements. |
| Misc. | Execution Plan | This event shows the plan tree of the Transact-SQL statement that was executed. |
| Misc. | Prepare SQL | This event indicates that an ODBC, OLE DB, or DB-Library application prepared a Transact-SQL statement, or statements, for use. |
| Misc. | Unprepare SQL | This event indicates that an ODBC, OLE DB, or DB-Library application unprepared a Transact-SQL statement, or statements, for use. |
| Sessions | Connect | This event indicates that a new connection has been made. |
| Sessions | Disconnect | This event indicates that a client has disconnected. |
| Sessions | Existing Connection | This event indicates that a connection existed when the SQL Profiler trace was started. |
| Stored Procedures | SP: Completed | This event indicates when a stored procedure has completed execution. |
| Stored Procedures | SP: Recompile | This event indicates that a stored procedure was recompiled during execution. |
| Stored Procedures | SP: Starting | This event indicates when a stored procedure has started execution. |
| Stored Procedures | SP: StmtCompleted | This event indicates when a statement in a stored procedure has completed execution. |
| TSQL: | SQL:BatchCompleted | This event indicates that a Transact-SQL batch completed. The **Text** column shows the statement that was executed. |
| TSQL: | SQL:StmtCompleted | This event indicates a Transact-SQL statement completed. The **Text** column shows the statement that was executed. |
| TSQL: | RPC:Completed | This event indicates that a remote procedure call (RPC) has completed. |

If your application is receiving timeout errors, stops responding (hangs), or experiences other events that cause the problem statements to never complete, also include the following events:

|  |  |  |
| --- | --- | --- |
| TSQL: | SQL:BatchStarting | This event indicates the start of a Transact-SQL batch. The **Text** column shows the statement being executed. |
| TSQL: | SQL:StmtStarting | This event indicates the start of a Transact-SQL statement. The **Text** column shows the statement being executed. |
| TSQL: | RPC:Starting | This event indicates the start of a remote procedure call (RPC). |
| Stored Procedures | SP: StmtStarting | This event indicates when a statement in a stored procedure is starting execution. |

This will help to make certain that you can see the statement that was executing when the timeout occurred

On the **Data Columns** tab, make sure that the following columns are included:

**For SQL Server 2000:** Start Time, End Time, LoginSid, SPID, Event Class, TextData, IntegerData, BinaryData, Duration, CPU, Reads, Writes, Application Name, NT User Name, DBUserName

**For SQL Server 7.0,** Start Time, End Time, Connection ID, SPID, Event Class, Text, Integer Data, Binary Data, Duration, CPU, Reads, Writes, Application Name, NT User Name, SQL User Name

For information about using SQL Profiler, see SQL Server 7.0 and SQL Server 2000 Books Online.   
  
2. Use Performance Monitor to capture Windows NT and SQL Server counters. To do so, follow these steps:

1. Start Windows NT Performance Monitor.
2. On the **View** menu, click **Log**.
3. On the **Options** menu, click **Log**.
4. Specify a file name and location to log the performance counters. You can adjust the update interval as appropriate.
5. On the **Edit** menu, click **Add To Log**.
6. Add all objects. Both the Windows NT and the SQL Server objects.
7. To start the log, on the **Options** menu, click **Log**, and then click the **Start Log** button.

For additional information, click the following article number to view the article in the Microsoft Knowledge Base:

[150934](http://support.microsoft.com/kb/150934/EN-US) How to Create a Performance Monitor Log for NT Troubleshooting

3. Check for blocking.  
  
To see if blocking is occurring, run the **sp\_who** system stored procedure:

exec sp\_who

This output will contain a **blk** column. Examine the output for any non-zero entries that indicate that blocking is occurring. Run this procedure periodically throughout the timeframe when the performance slowdown is occurring.   
  
**Note** Running the **sp\_who** system stored procedure is a check to see if blocking exists. Typically, it is not enough information to fully troubleshoot a blocking problem. For additional information, click the following article number to view the article in the Microsoft Knowledge Base:

[251004](http://support.microsoft.com/kb/251004/EN-US) INF: How to Monitor SQL Server 7.0 Blocking

#### Run the Application Under Typical Load

Ideally, it is best to capture the SQL Profiler, Performance Monitor, and blocking output during the same timeframe. This timeframe must encompass a time when application performance goes from good to bad. The combination of this information will help you to get a clearer picture of where the performance slowdown is occurring.

#### Interpret the Results

1. Check for blocking.   
     
   If the **blk** column in the **sp\_who** output is non-zero, this indicates that blocking is occurring on your system. If processes are blocking each other, the processes that are being blocked can experience longer execution times. For additional information, click the following article number to view the article in the Microsoft Knowledge Base:

[224453](http://support.microsoft.com/kb/224453/EN-US) INF: Understanding and Resolving SQL Server 7.0 or 2000 Blocking Problems

1. Examine the SQL Profiler output.   
     
   Viewing SQL Profiler data efficiently is extremely valuable in resolving performance issues. The most important thing to realize is that you do not have to look at everything you captured. Be selective. SQL Profiler provides capabilities to help you effectively view the captured data. On the **Properties** tabs (click **Properties** on the **File** menu), SQL Profiler allows you to limit the data displayed by removing data columns or events, grouping (sorting) by data columns, and applying filters. You can search the whole trace or only a specific column for specific values (on the **Edit** menu, click **Find**). You can also save the SQL Profiler data to a SQL Server table (on the **File** menu, point to **Save As**, click **Trace Table**), and then run SQL queries against it.   
     
   Be careful that you perform filtering only on a previously saved trace file. If you perform these steps on an active trace, you risk losing data that has been captured since the trace was started. Save an active trace to a file or table first (on the**File** menu, click **Save As**) and then reopen it (on the **File** menu, click **Open**) before you continue. When you work on a saved trace file, the filtering does not permanently remove the data that is being filtered out; it just does not display it. You can add and remove events and data columns as needed to help focus your searches.   
     
   The first step in examining SQL Profiler trace files for performance cases is to determine where the different types of events are occurring on the server.   
     
   **Group the trace by Event Class**:

a. On the **File** menu, click **Properties**.   
  
b. On the **Data Columns** tab, use the UP button to move **Event Class** under the **Groups** heading, and the DOWN button to remove all other columns under the **Groups** heading.   
  
c. Click **OK**.

Grouping by the event class column shows what type of events are occurring on SQL Server, and how frequently. Search this column for the following events:  
  
**SP:RECOMPILE**

This event indicates that a stored procedure was recompiled during execution. Several recompile events indicate that SQL Server is spending resources on query compilation instead of query execution.   
  
For additional information about troubleshooting stored procedure recompilations, click the following article number to view the article in the Microsoft Knowledge Base:

[243586](http://support.microsoft.com/kb/243586/EN-US) INF: Troubleshooting Stored Procedure Recompilation

**Attention**

An attention signal indicates that a query was canceled by a client. This is generally because of one of two causes:  
The user explicitly canceled the query or ended the application.  
  
-or-  
  
A query timeout was exceeded.   
  
If you see attention signals, this may indicate that certain queries are running slowly.   
  
For additional information, click the following article number to view the article in the Microsoft Knowledge Base:

[243589](http://support.microsoft.com/kb/243589/EN-US) HOW TO: Troubleshoot Slow-Running Queries on SQL Server 7.0 or Later

To help identify the query that received the attention signal, revise the trace so that it is not grouped by any data column, and filter on the system process ID (SPID) that received it (on the **Filters** tab, set SPID = *x*). The**SQL:StmtStarting**, **SQL:BatchStarting**, or **SP:StmtStarting** event immediately preceding the attention signal is the query that received the timeout or cancel. You can search the **Event Class** column for the Attention event to easily locate it (on the **Edit** menu, click **Find**).

**PREPARE SQL and EXEC PREPARED SQL**

The **Prepare SQL** event indicates that an ODBC, OLE DB, or DB-Library application prepared a Transact-SQL statement ,or statements, for use. The **Exec Prepared SQL** event indicates that the application made use of an existing prepared statement to run a command.   
  
Compare the number of times these two events occur. Ideally, an application must prepare a SQL statement one time and run it several times. This saves the Optimizer the cost of compiling a new plan each time the statement is executed. Therefore, the number of **Exec Prepared SQL**events should be much larger than the number of**Prepare SQL** events. If the number of **Prepare SQL** events is roughly equivalent to the number of **Exec Prepared SQL**events, this may indicate that the application is not making good use of the prepare/execute model. It is best not to prepare a statement that is only going to be executed a single time. For more information about preparing SQL statements, see the "Preparing SQL Statements" topic in SQL Server 7.0 Books Online.   
  
If the number of **Exec Prepared SQL** events is not three to five times greater than the number of **Prepare SQL**events, the application may not be making efficient use of the prepare/execute model. For additional information, click the following article number to view the article in the Microsoft Knowledge Base:

[243588](http://support.microsoft.com/kb/243588/EN-US) HOW TO: Troubleshoot the Performance of Ad-Hoc Queries

In SQL Server 2000, the excessive roundtrips per prepare/execute will be eliminated, so the 3-5 ratio is not as stringent. However, it can still be a good rule, to try and reuse the prepared plan more than one time.

**Missing Column Statistics**

This event indicates that statistical information the Optimizer could have used to generate a better query plan was unavailable. This indicates that the query does not have useful indexes on at least one table involved. Beyond not having a useful index, SQL Server does not even have statistical data about the columns involved to make an informed decision for a query plan. The outcome is that the query plan generated may not be the optimal one. If you see these events, look at the query and the execution plan generated, and then see the following article in the Microsoft Knowledge Base for steps to take to improve the performance of this query:

[243589](http://support.microsoft.com/kb/243589) HOW TO: Troubleshoot Slow-Running Queries on SQL Server 7.0 or Later

When you view the **Missing Column Statistics** events, focus first on those that occur in association with long-running queries. Some events may be generated and resolved automatically by SQL Server with autostats and may not require user intervention. Therefore, the best strategy is to first focus on queries with long duration, as shown later in this article, and note if there are associated **Missing Column Statistics** events.  
  
If you are not seeing instances of these event classes, the next step is to determine where the time is being spent.  
  
**Group the trace output by Duration**:

a. On the **File** menu, click **Properties**.   
  
On the **Data Columns** tab, use the UP button to move **Duration** under the **Groups** heading, and the DOWN button to remove all other columns under the **Groups**heading.   
  
c. On the **Events** tab, remove all groups except **TSQL** and **Stored Procedures**.  
  
d. Click **OK**.

By grouping on duration, you can easily see what SQL statements, batches, or procedures are running the slowest. It is very important to look not only at the time when the problem is occurring, but also to get a baseline of when performance is good to compare against. You can filter on start time to break the trace up into sections when performance was good, and a separate section for when performance was poor. Look for the queries with the longest duration when performance is good. These are most likely the root of the problem. If overall system performance is decreased, even good queries can show long durations as they are waiting on system resources.   
  
If you see a small number of queries with high durations, see the following article in the Microsoft Knowledge Base:

[243589](http://support.microsoft.com/kb/243589) HOW TO: Troubleshoot Slow-Running Queries on SQL Server 7.0 or Later

If you see that the duration of individual queries is low, but there are several of them, and the **SQL Compilations/sec** counter in Performance Monitor output (described later) is high, see the following article in the Microsoft Knowledge Base:

[243588](http://support.microsoft.com/kb/243588) HOW TO: Troubleshoot the Performance of Ad-Hoc Queries

**Examine the remaining data columns**:  
  
Additional insight into the nature of a performance problem can be gained by viewing the other data columns in the trace data. Here are some things to consider:

If the CPU usage is high, group by CPU to see what queries are the biggest users of CPU time. Search the **Text** column for "hash" or "merge" to find what query execution plan is using these join types. They are more CPU and memory intensive than a nested loop join, which is generally IO intensive.  
  
If disk IO is the bottleneck, group by reads and writes. View the **Application Name**, **NT User Name**, and **SQL User Name** fields to help isolate the source of a long-running query.   
  
The integer data column of the exception event will indicate any errors that were returned back to the client. You can find the text of the error message by searching on the number in SQL Server 7.0 Books Online.  
  
The **Connection ID** field is helpful to make sure that you are looking at the same sessions for a specific client. A SPID cannot guarantee this, as a user may have disconnected and a new user connected and received the same SPID.

The benefit derived from these fields may vary depending on the scenario, but they should be examined if the obvious fields earlier in this article do not provide an answer.

1. Examine the Performance Monitor output.   
     
   Performance Monitor will show you the overall system bottlenecks. It may be that SQL Server and the application are performing as expected, but the computer is underpowered, lacking memory or other resources. Or certain counters can indicate problems with the way the application and SQL Server are performing. At a minimum, check the following counters:

* Object: Process  
  Counter: Processor  
  Instance: SQL Server
* Object: Processor  
  Counter: %Processor Time  
  Instance: Check each processor instance
* Object: Physical Disk  
  Counter: Avg. Disk Queue Length  
  Instance: Check each physical disk instance
* Object: SQL Server:SQL Statistics  
  Counter: SQL Compilations/sec

Look for a trend over the timeframe from when performance went from good to bad: what increased first? Is the computer CPU bound or DISK IO bound? This information, together with the Profiler output earlier in this article, will help you narrow down the problem areas. High CPU problems may indicate large numbers of stored procedure recompilations, ad-hoc query compilations, or intensive use of hash and merge joins. The articles referenced earlier in this article must be followed to determine the correct course of action. High disk queue lengths may indicate the need for more system memory or an improved disk subsystem.

# SQL Server Performance Tuning

SQL Server Performance Tuning and health check is a very challenging subject that requires expertise in Database Administration and Database Development. Here are few pointers how one can keep their SQL Server Performance Optimal. I am often asked what can one do keep SQL Server Health Optimal and SQL Server keep on running very smooth.

Here is the quick list of the pointers which one should consider for performance tuning.

### ****Server/Instance Level Configuration Check****

Review all the SQL Server/Instance Level settings of the server and tune it based on system workload.

### ****I/O distribution Analysis****

Analyze the I/O of your system and decide the best distribution for the I/O load. Perform object level analysis and do performance tuning at table level. Reduce I/O performance bottlenecks and suggest optimal setting for read and write database. This is especially critical for databases that need to sustain heavy updates during peak usage hours.

### ****SQL Server Resource Wait Stats Analysis****

Wait Stat Analysis is very crucial for optimizing databases, but is often overlooked due to lack of understanding. Perform advanced resource wait statistics analysis to proactively reduce performance bottleneck.

### ****Index Analysis****

Indexes are considered valuable for performance improvements. Analyze all your indexes to identify non-performing indexes and missing indexes that can improve performance.

### ****TempDB Space Review****

Review the size and usage of your TempDB database.

### ****Database Files (MDF, NDF) and Log File Inspection****

Review all the files and filegroups of each of your databases and analysis them to identify any object or files that are causing bottlenecks.

### ****Fragmentations and Defragmentations****

Identify the optimal settings of your database files and tables to reduce fragmentation and reduce them.

### ****Backup and Recovery health Check****

Review all backup & recovery settings and procedures of your databases and understand the functions of the respective databases.

### ****Log Reviews (Windows Event Logs, SQL Server Error Logs and Agent Logs)****

Logs reveal many hidden facts. Identity the critical errors and resolve them.

### ****Hardware Review****

Review the hardware and and verify that it positively impact the performance or scalability of the database.

### ****DBCC Best Practices Implementations****

There are few DBCC commands to be avoided and few very crucial for system. Understand the usage of DBCC FREEPROCCACHE, DBCC SRHINKDATABASE, DBCC SHRINKFILE, DBCC DROPCLEANBUFFER, DBCC REINDEX, as well as the usage of few system stored procedures like SP\_UPDATESTATS. If you are currently using any of the above mentioned and a few other DBCC maintenance task commands carefully review their usage.

### ****Deadlock Detection and Resolutions Hands-On****

Detecting deadlock is not very difficult, but to learn the tips and trick to resolve them requires an understanding of the issue and some experience. Understand the locking mechanism and resolve them.

pinal at SQLAuthority.com