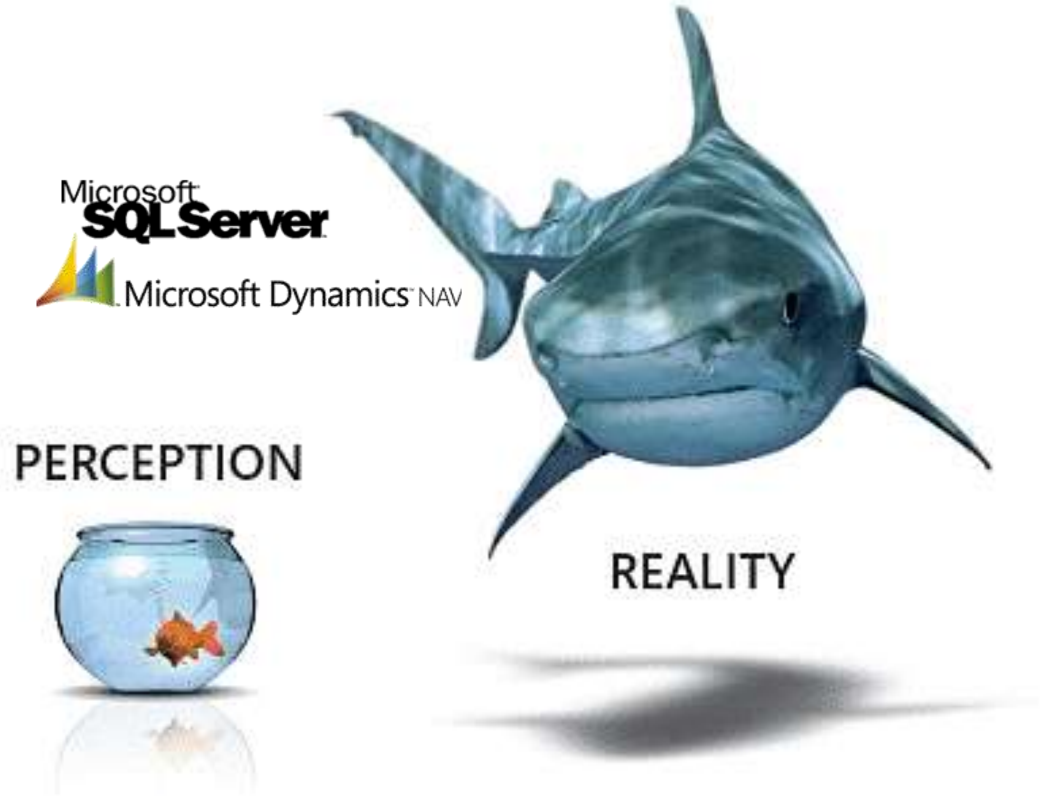
Microsoft SQL Server

Microsoft Dynamics NAV



Introduction Jörg Stryk

MS Dynamics NAV (Navision) since 1997
(MCBMSS)

MS SQL Server since 2003
(MCP, MCPS)

Focus on „***NAV/SQL Performance Optimization***“
STRYK System Improvement (since 2006, Certified Dynamics NAV Solution Provider)
Worldwide support of MS Dynamics Partners & Customers

Microsoft Most Valuable Professional
(MVP MS Dynamics NAV)

Book:	*“NAV/SQL Performance Field Guide”*	(ISBN 978-3-8370-1442-6)
Software:	*“NAV/SQL Performance Toolbox”*	http://www.stryk.info/toolbox.html
Blog:	*“NAV/SQL Performance – My Two Cents”*	http://dynamicsuser.net/blogs/stryk/
MSDynamicsWorld.com:	*“NAV/SQL Quickies”*	http://msdynamicsworld.com/column/sql-server/nav-sql-quickie

Contact: contact@stryk.info
http://www.stryk.info

DECISIONS
SPRING 2010

Agenda

NAV/SQL Performance Optimization:

Automatic Block and Deadlock Detection

Abstract:

Blocking and **Deadlocking** unfortunately are major problems with NAV & SQL Server:
Too **long response times**, **timeouts** and **killed processes** can dramatically slow down the user's daily work. But as blocking conflicts are – besides the program code - a matter of workflow, timing and probability it is hardly possible to **predict** those issues during development. So blocks & deadlocks are mostly to be encountered in real life scenarios, hence, it is crucial to **quickly** implement appropriate solutions.

The first step in solving these issues is to thoroughly **measure** and **investigate** them - to identify, quantify and qualify the problems; then sufficient **solutions** could be developed and deployed.

This session will show how blocks & deadlocks could be efficiently tracked and analyzed with commonly available out-of-the box features of SQL Server.

Download this presentation and script templates here:

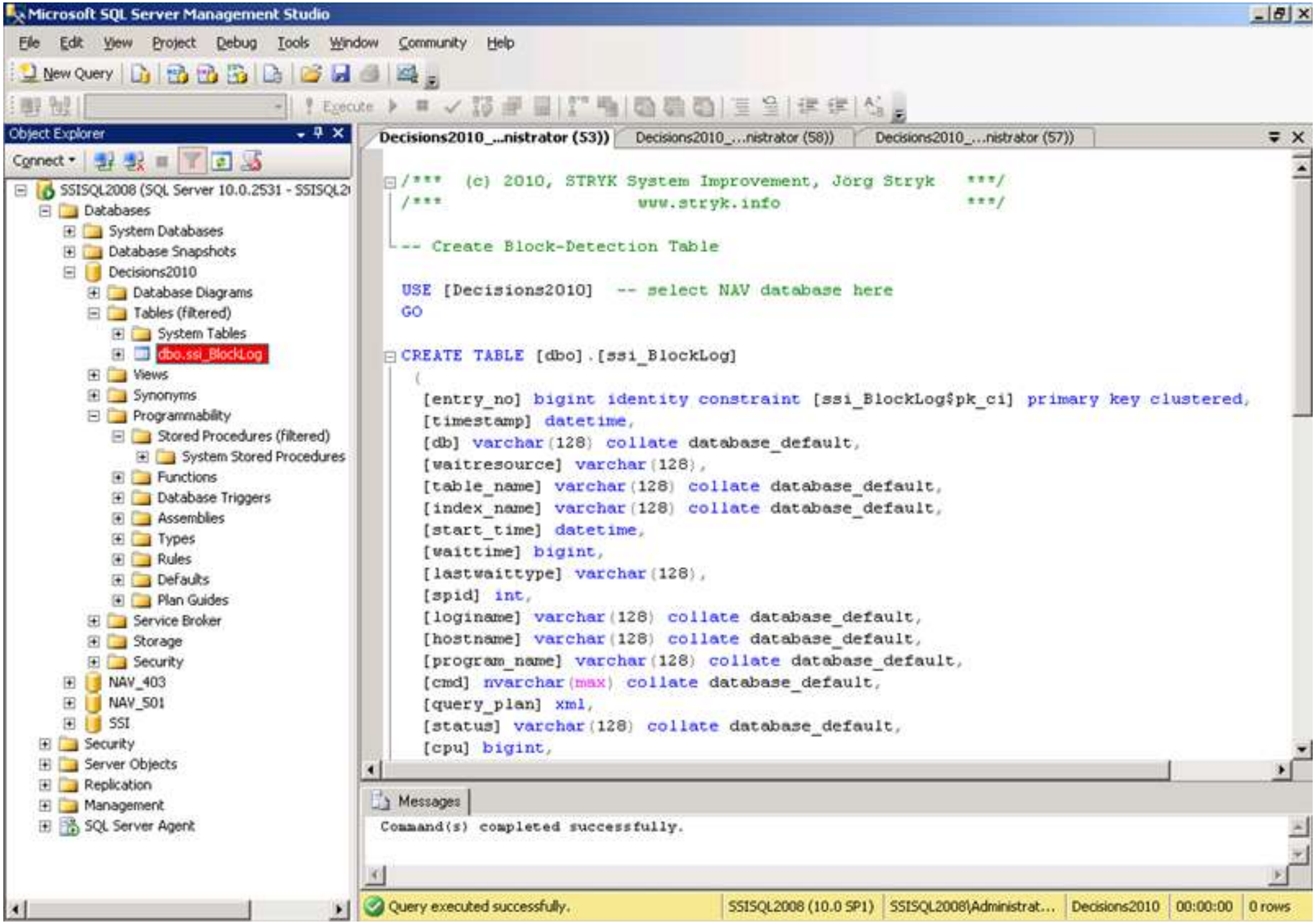
http://dynamicsuser.net/blogs/stryk/archive/2010/05/19/decisions-spring-2010-nav-sql-performance-blocks-and-deadlocks.aspx

Detecting Blocks

Step 1: Create “container” **table** to store recorded block information
Step 2: Create **procedure** to gather and store block data
Step 3: Create SQL Server Agent **Job** for recording
Step 4: Create SQL Server **Alert** to trigger the recording
Step 5: Block **Analysis**

DECISIONS SPRING 2010

Step 1: Create “container” **table** to store recorded block information

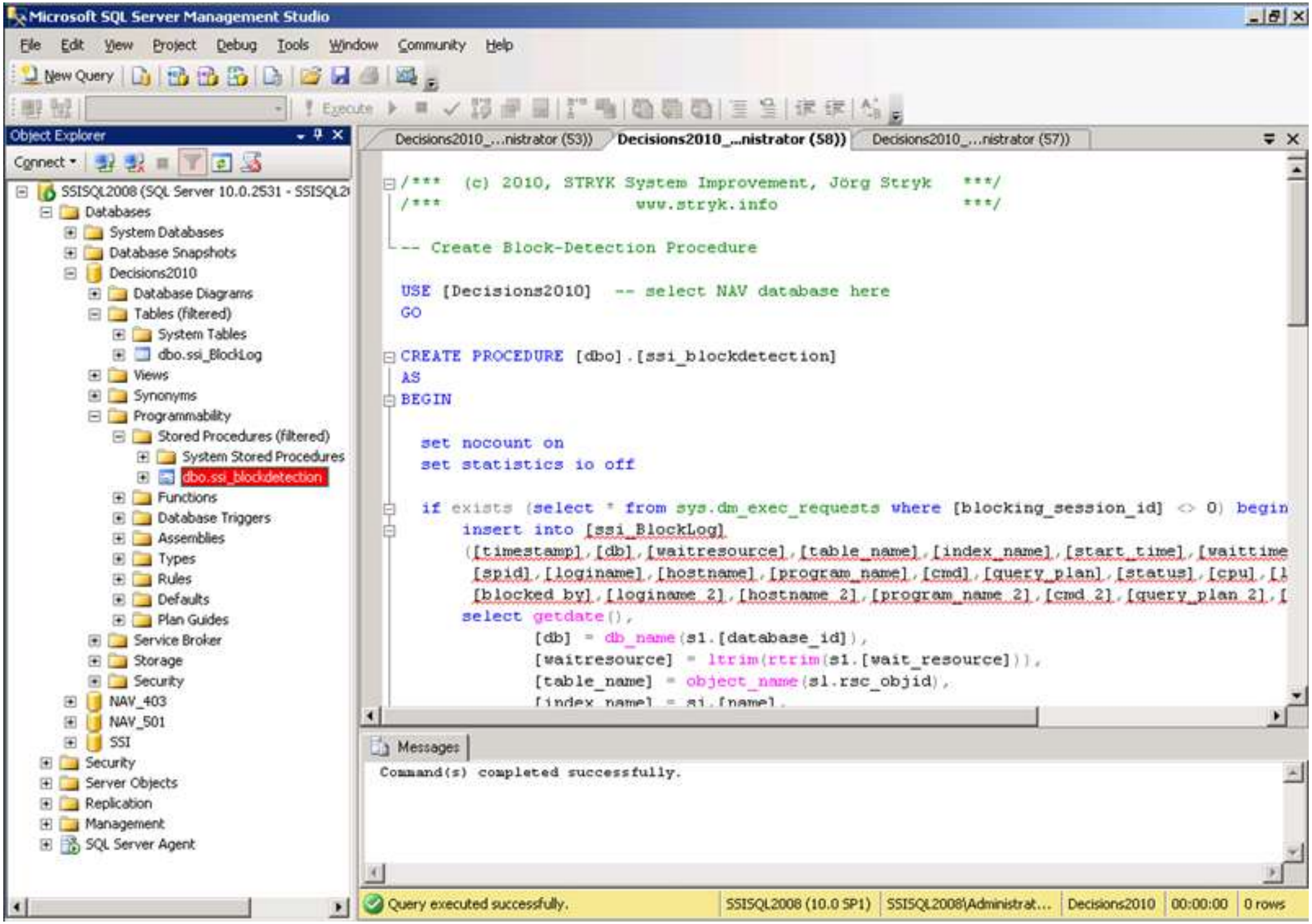


```
/***  (c) 2010, STRYK System Improvement, Jörg Stryk   ***/
/***                 www.stryk.info                    ***/

-- Create Block-Detection Table

USE [Decisions2010]  -- select NAV database here
GO

CREATE TABLE [dbo].[ssi_BlockLog]
 (
  [entry_no] bigint identity constraint [ssi_BlockLog$pk_ci] primary key clustered,
  [timestamp] datetime,
  [db] varchar(128) collate database_default,
  [waitresource] varchar(128),
  [table_name] varchar(128) collate database_default,
  [index_name] varchar(128) collate database_default,
  [start_time] datetime,
  [waittime] bigint,
  [lastwaittype] varchar(128),
  [spid] int,
  [loginame] varchar(128) collate database_default,
  [hostname] varchar(128) collate database_default,
  [program_name] varchar(128) collate database_default,
  [cmd] nvarchar(max) collate database_default,
  [query_plan] xml,
  [status] varchar(128) collate database_default,
  [cpu] bigint,
```

```
Command(s) completed successfully.
```

Query executed successfully. | SSISQL2008 (10.0 SP1) | SSISQL2008\Administrat... | Decisions2010 | 00:00:00 | 0 rows

Step 2: Create **procedure** to gather and store block data

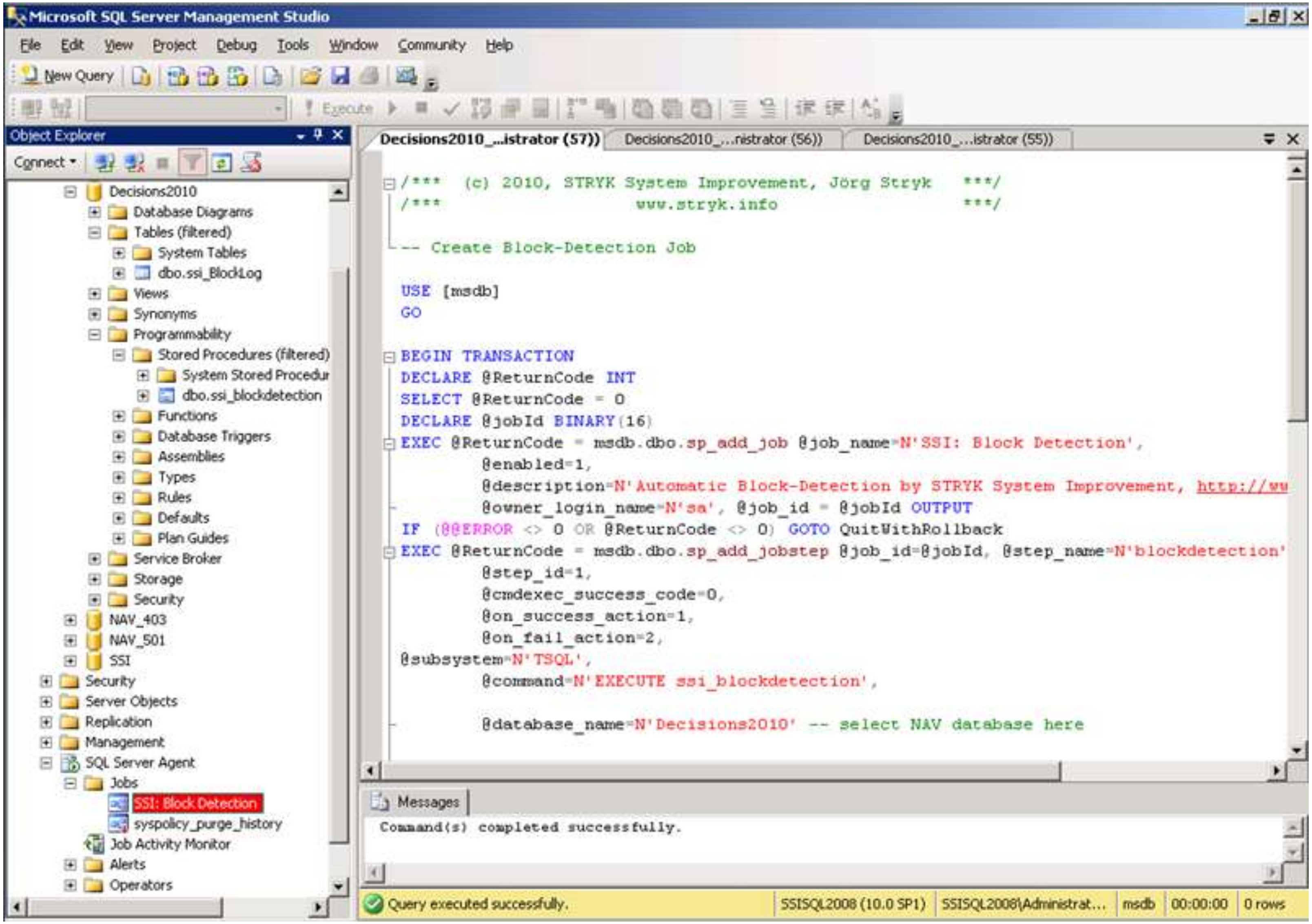


```
/***  (c) 2010, STRYK System Improvement, Jörg Stryk   ***/
/***                  www.stryk.info                    ***/

-- Create Block-Detection Procedure

USE [Decisions2010]  -- select NAV database here
GO

CREATE PROCEDURE [dbo].[ssi_blockdetection]
AS
BEGIN

  set nocount on
  set statistics io off

  if exists (select * from sys.dm_exec_requests where [blocking_session_id] <> 0) begin
      insert into [ssi_BlockLog]
      ([timestamp],[db],[waitresource],[table_name],[index_name],[start_time],[waittime
       [spid],[loginame],[hostname],[program_name],[cmd],[query_plan],[status],[cpu],[l
       [blocked_by],[loginame_2],[hostname_2],[program_name_2],[cmd_2],[query_plan_2],[
      select getdate(),
             [db] = db_name(s1.[database_id]),
             [waitresource] = ltrim(rtrim(s1.[wait_resource])),
             [table_name] = object_name(s1.rsc_objid),
             [index_name] = si.[name],
```

```
Command(s) completed successfully.
```

Query executed successfully. | SSISQL2008 (10.0 SP1) | SSISQL2008\Administrat... | Decisions2010 | 00:00:00 | 0 rows

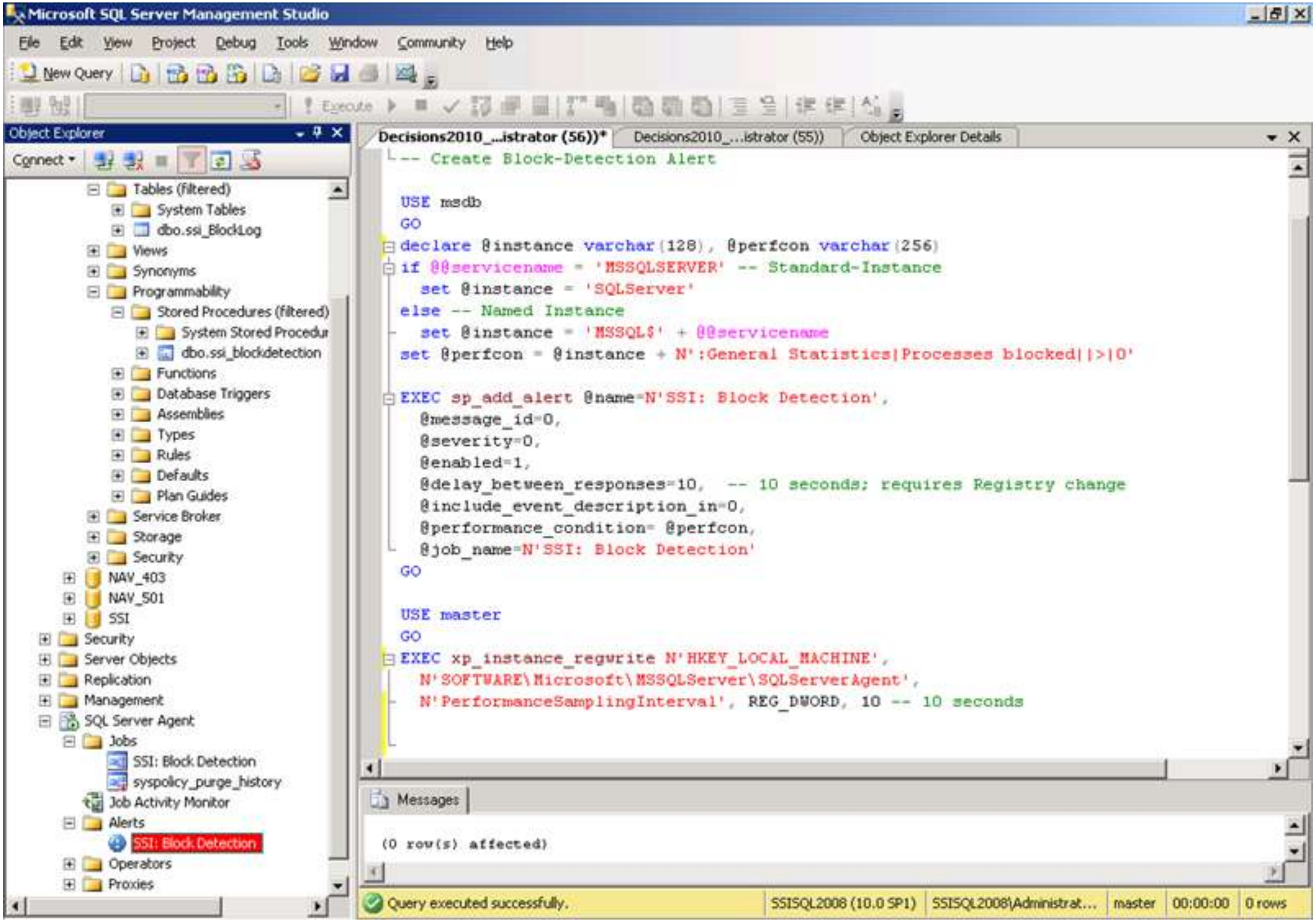
Step 3: Create SQL Server Agent **Job** for recording



```sql
/***  (c) 2010, STRYK System Improvement, Jörg Stryk   ***/
/***                   www.stryk.info                  ***/

-- Create Block-Detection Job

USE [msdb]
GO

BEGIN TRANSACTION
DECLARE @ReturnCode INT
SELECT @ReturnCode = 0
DECLARE @jobId BINARY(16)
EXEC @ReturnCode = msdb.dbo.sp_add_job @job_name=N'SSI: Block Detection',
        @enabled=1,
        @description=N'Automatic Block-Detection by STRYK System Improvement, http://ww
        @owner_login_name=N'sa', @job_id = @jobId OUTPUT
IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback
EXEC @ReturnCode = msdb.dbo.sp_add_jobstep @job_id=@jobId, @step_name=N'blockdetection'
        @step_id=1,
        @cmdexec_success_code=0,
        @on_success_action=1,
        @on_fail_action=2,
@subsystem=N'TSQL',
        @command=N'EXECUTE ssi_blockdetection',

        @database_name=N'Decisions2010' -- select NAV database here
```

```
Command(s) completed successfully.
```

Step 4: Create SQL Server **Alert** to trigger the recording



```sql
-- Create Block-Detection Alert

USE msdb
GO
declare @instance varchar(128), @perfcon varchar(256)
if @@servicename = 'MSSQLSERVER' -- Standard-Instance
  set @instance = 'SQLServer'
else -- Named Instance
  set @instance = 'MSSQL$' + @@servicename
set @perfcon = @instance + N':General Statistics|Processes blocked||>|0'

EXEC sp_add_alert @name=N'SSI: Block Detection',
  @message_id=0,
  @severity=0,
  @enabled=1,
  @delay_between_responses=10,  -- 10 seconds; requires Registry change
  @include_event_description_in=0,
  @performance_condition= @perfcon,
  @job_name=N'SSI: Block Detection'
GO

USE master
GO
EXEC xp_instance_regwrite N'HKEY_LOCAL_MACHINE',
  N'SOFTWARE\Microsoft\MSSQLServer\SQLServerAgent',
  N'PerformanceSamplingInterval', REG_DWORD, 10 -- 10 seconds
```

```
(0 row(s) affected)
```

Query executed successfully. | SSISQL2008 (10.0 SP1) | SSISQL2008\Administrat... | master | 00:00:00 | 0 rows

Detecting Blocks

How it works …

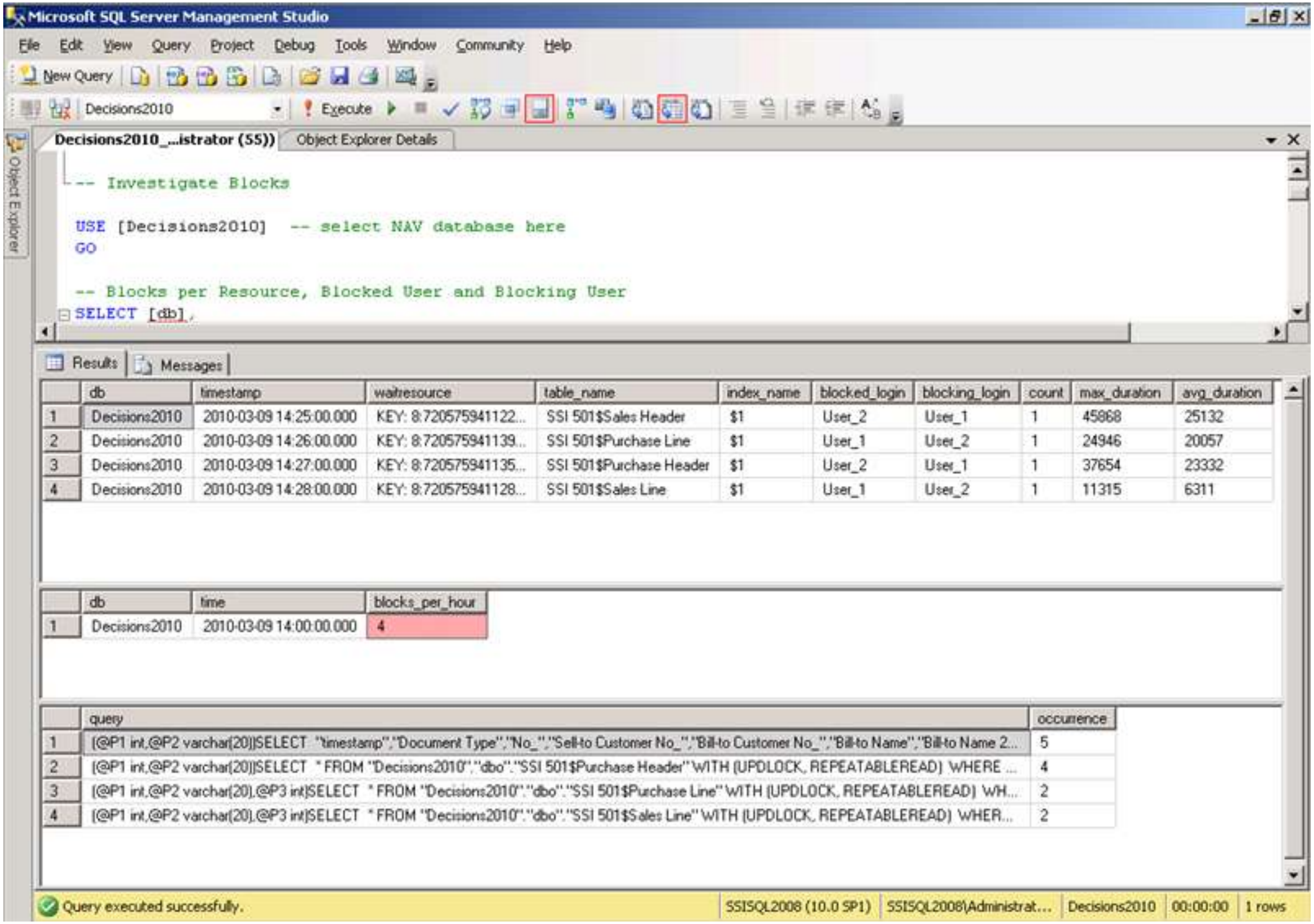
Whenever a process gets blocked the *Performance Counter* "**SQL Server: General Statistics – Processes blocked**" will count them.
If this number raises above the value 0 (zero) the *Alert* "**SSI: Block Detection**" is triggered. The Alert responds with running the *Job* "**SSI: Block Detection**" which executes the *Stored Procedure* "**ssi_blockdetection**".
This SP collects all relevant information about the block – *who is blocking whom, where, when and how* - and saves this into *Table* "**ssi_BlockLog**".

Hence, a fully automatic and event-triggered block detection process is established!

Then the recorded data could be **analyzed** to determine frequent/recurring problems, affected queries and users and much more.

DECISIONS
SPRING 2010

Step 5: Block **Analysis**


Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Community Help
New Query
Decisions2010
Execute
Decisions2010_...istrator (55))
Object Explorer Details
Object Explorer
Results
Messages

```sql
-- Investigate Blocks

USE [Decisions2010]  -- select NAV database here
GO

-- Blocks per Resource, Blocked User and Blocking User
SELECT [db],
```

	db	timestamp	waitresource	table_name	index_name	blocked_login	blocking_login	count	max_duration	avg_duration
1	Decisions2010	2010-03-09 14:25:00.000	KEY: 8:720575941122...	SSI 501$Sales Header	$1	User_2	User_1	1	45868	25132
2	Decisions2010	2010-03-09 14:26:00.000	KEY: 8:720575941139...	SSI 501$Purchase Line	$1	User_1	User_2	1	24946	20057
3	Decisions2010	2010-03-09 14:27:00.000	KEY: 8:720575941135...	SSI 501$Purchase Header	$1	User_2	User_1	1	37654	23332
4	Decisions2010	2010-03-09 14:28:00.000	KEY: 8:720575941128...	SSI 501$Sales Line	$1	User_1	User_2	1	11315	6311

	db	time	blocks_per_hour
1	Decisions2010	2010-03-09 14:00:00.000	4

	query	occurrence
1	(@P1 int,@P2 varchar(20))SELECT "timestamp","Document Type","No_","Sell-to Customer No_","Bill-to Customer No_","Bill-to Name","Bill-to Name 2...	5
2	(@P1 int,@P2 varchar(20))SELECT * FROM "Decisions2010"."dbo"."SSI 501$Purchase Header" WITH (UPDLOCK, REPEATABLEREAD) WHERE ...	4
3	(@P1 int,@P2 varchar(20),@P3 int)SELECT * FROM "Decisions2010"."dbo"."SSI 501$Purchase Line" WITH (UPDLOCK, REPEATABLEREAD) WH...	2
4	(@P1 int,@P2 varchar(20),@P3 int)SELECT * FROM "Decisions2010"."dbo"."SSI 501$Sales Line" WITH (UPDLOCK, REPEATABLEREAD) WHER...	2

Query executed successfully. | SSISQL2008 (10.0 SP1) | SSISQL2008\Administrat... | Decisions2010 | 00:00:00 | 1 rows

Detecting Deadlocks

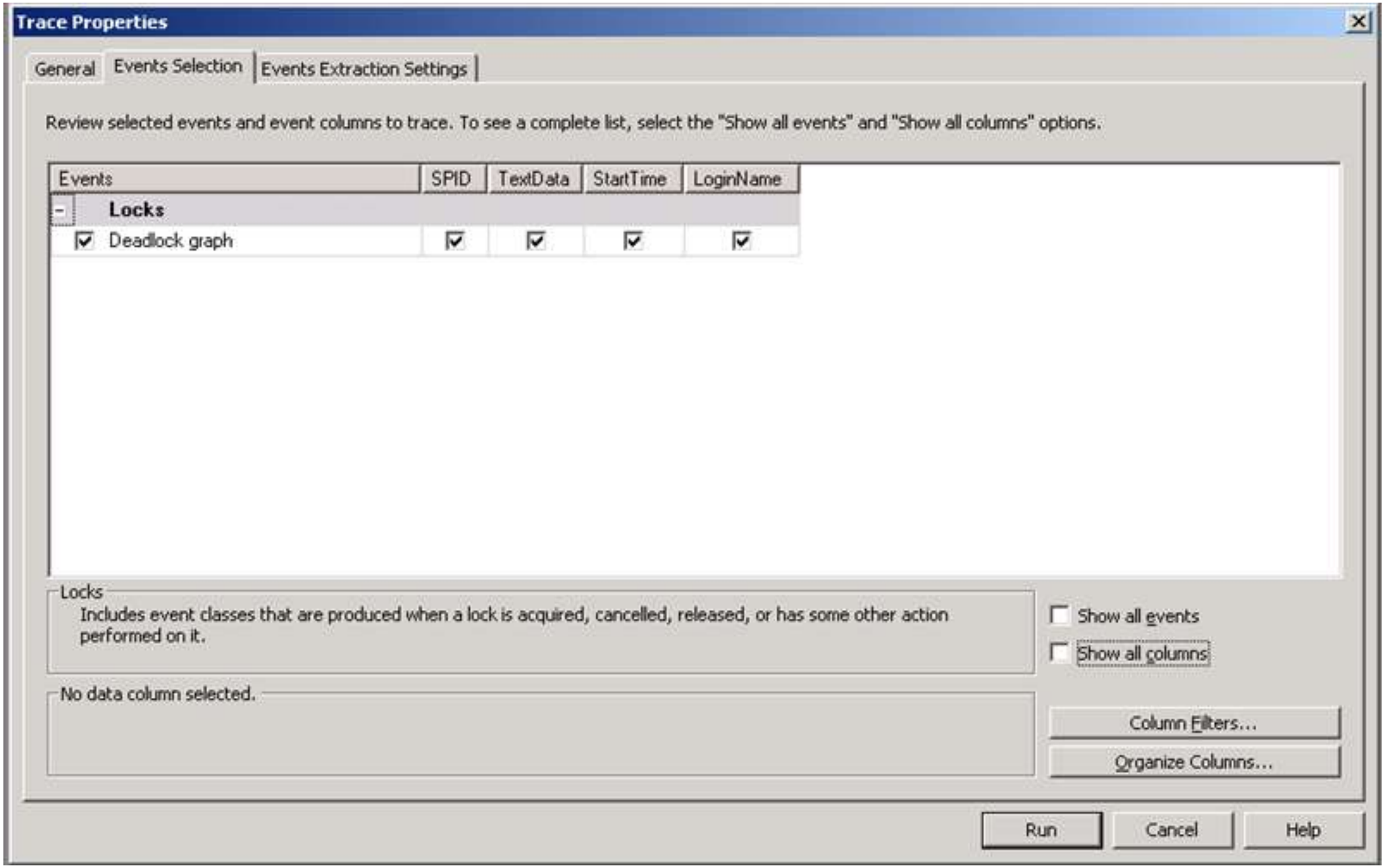
Step 1: Create SQL Profiler **Trace** for Deadlocks

Step 2: Create SQL Server Agent **Job** to start trace

Step 3: Extract “**Deadlock Graphs**”

Step 4: Deadlock **Analysis**

Step 1: Create SQL Profiler **Trace** for Deadlocks



Review selected events and event columns to trace. To see a complete list, select the "Show all events" and "Show all columns" options.

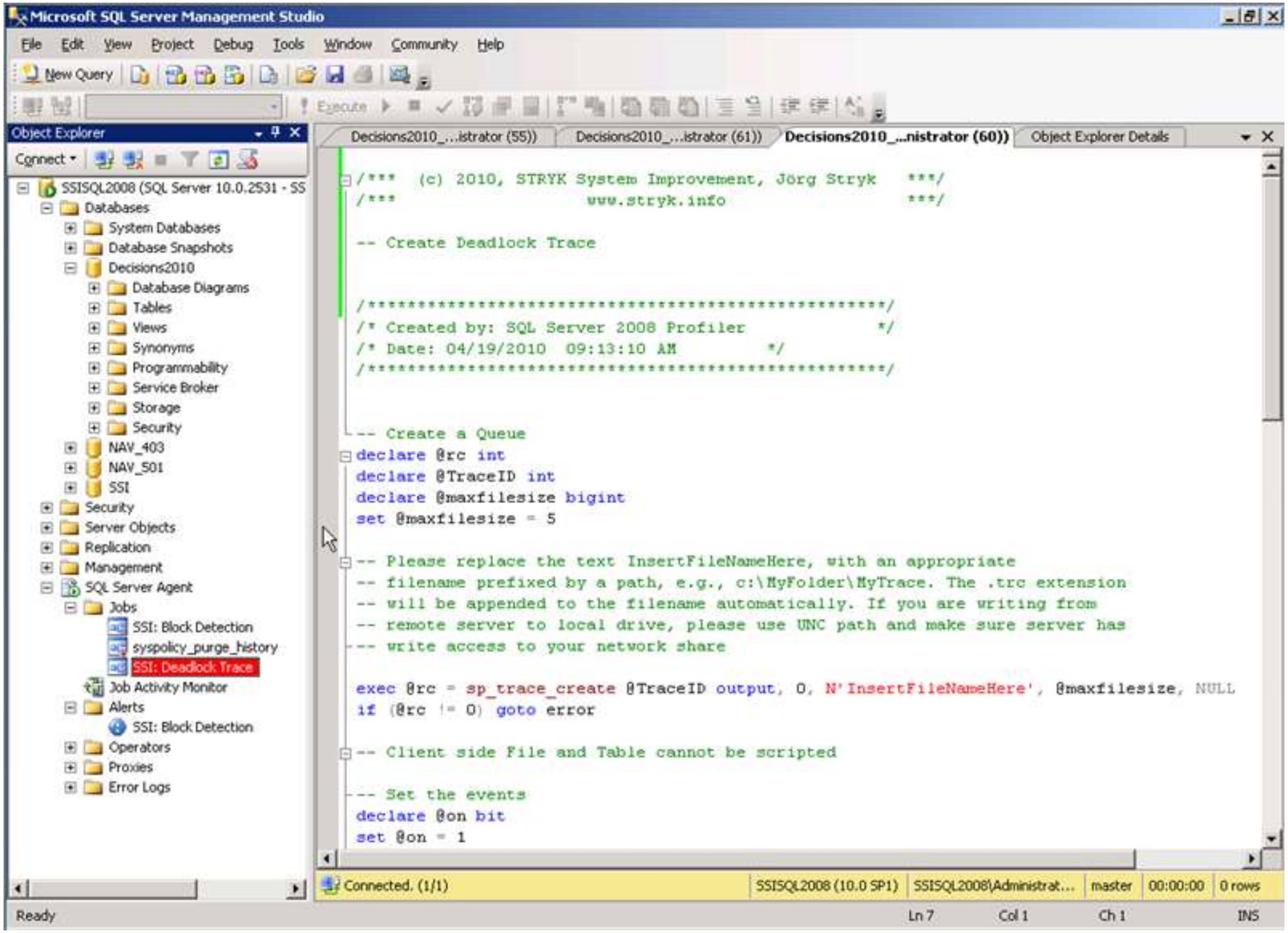
Events	SPID	TextData	StartTime	LoginName
− **Locks**				
☑ Deadlock graph	☑	☑	☑	☑

Locks
Includes event classes that are produced when a lock is acquired, cancelled, released, or has some other action performed on it.

No data column selected.

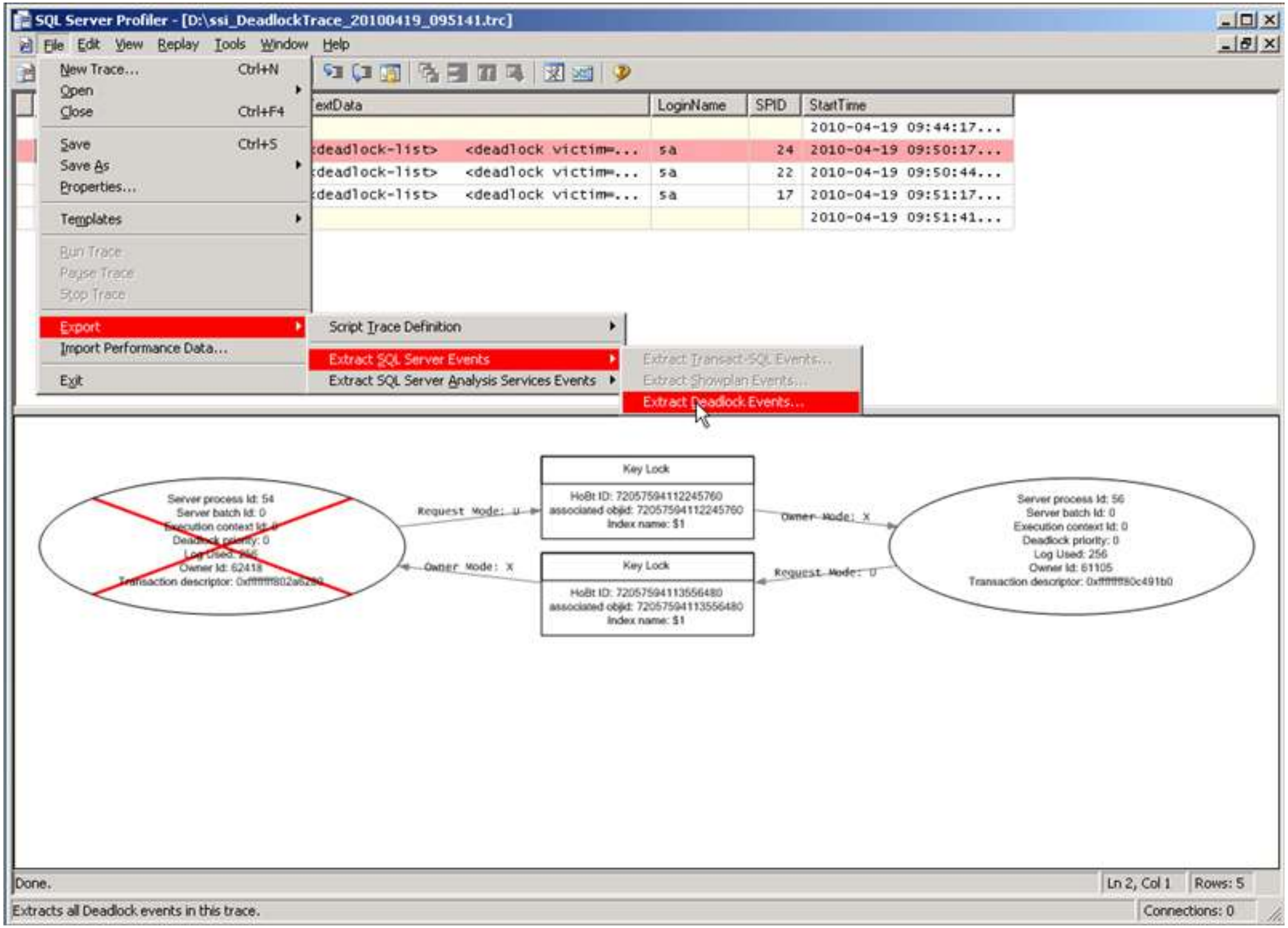
☐ Show all events
☐ Show all columns

Step 2: Create SQL Server Agent **Job** to start trace



```sql
/***  (c) 2010, STRYK System Improvement, Jörg Stryk   ***/
/***                  www.stryk.info                    ***/

-- Create Deadlock Trace


/****************************************************/
/* Created by: SQL Server 2008 Profiler             */
/* Date: 04/19/2010  09:13:10 AM         */
/****************************************************/


-- Create a Queue
declare @rc int
declare @TraceID int
declare @maxfilesize bigint
set @maxfilesize = 5

-- Please replace the text InsertFileNameHere, with an appropriate
-- filename prefixed by a path, e.g., c:\MyFolder\MyTrace. The .trc extension
-- will be appended to the filename automatically. If you are writing from
-- remote server to local drive, please use UNC path and make sure server has
-- write access to your network share

exec @rc = sp_trace_create @TraceID output, 0, N'InsertFileNameHere', @maxfilesize, NULL
if (@rc != 0) goto error

-- Client side File and Table cannot be scripted

-- Set the events
declare @on bit
set @on = 1
```

Step 3: Extract “**Deadlock Graphs**”


SQL Server Profiler - [D:\ssi_DeadlockTrace_20100419_095141.trc]
File Edit View Replay Tools Window Help
New Trace... Ctrl+N
Open
Close Ctrl+F4
Save Ctrl+S
Save As
Properties...
Templates
Run Trace
Pause Trace
Stop Trace
Export
Import Performance Data...
Exit
Script Trace Definition
Extract SQL Server Events
Extract SQL Server Analysis Services Events
Extract Transact-SQL Events...
Extract Showplan Events...
Extract Deadlock Events...
Server process Id: 54
Server batch Id: 0
Execution context Id: 0
Deadlock priority: 0
Log Used: 256
Owner Id: 62418
Transaction descriptor: 0xffffff802a6280
Request Mode: U
Owner Mode: X
Key Lock
HoBt ID: 72057594112245760
associated objid: 72057594112245760
Index name: $1
Key Lock
HoBt ID: 72057594113556480
associated objid: 72057594113556480
Index name: $1
Owner Mode: X
Request Mode: U
Server process Id: 56
Server batch Id: 0
Execution context Id: 0
Deadlock priority: 0
Log Used: 256
Owner Id: 61105
Transaction descriptor: 0xffffff80c491b0

extData	LoginName	SPID	StartTime
			2010-04-19 09:44:17...
deadlock-list> <deadlock victim=...	sa	24	2010-04-19 09:50:17...
deadlock-list> <deadlock victim=...	sa	22	2010-04-19 09:50:44...
deadlock-list> <deadlock victim=...	sa	17	2010-04-19 09:51:17...
			2010-04-19 09:51:41...

Done. Ln 2, Col 1 Rows: 5

Extracts all Deadlock events in this trace. Connections: 0

Detecting Deadlocks

How it works …

Using **SQL Profiler** a trace which records the "*Locks: Deadlock Graph*" event was assembled and exported as TSQL **script**.

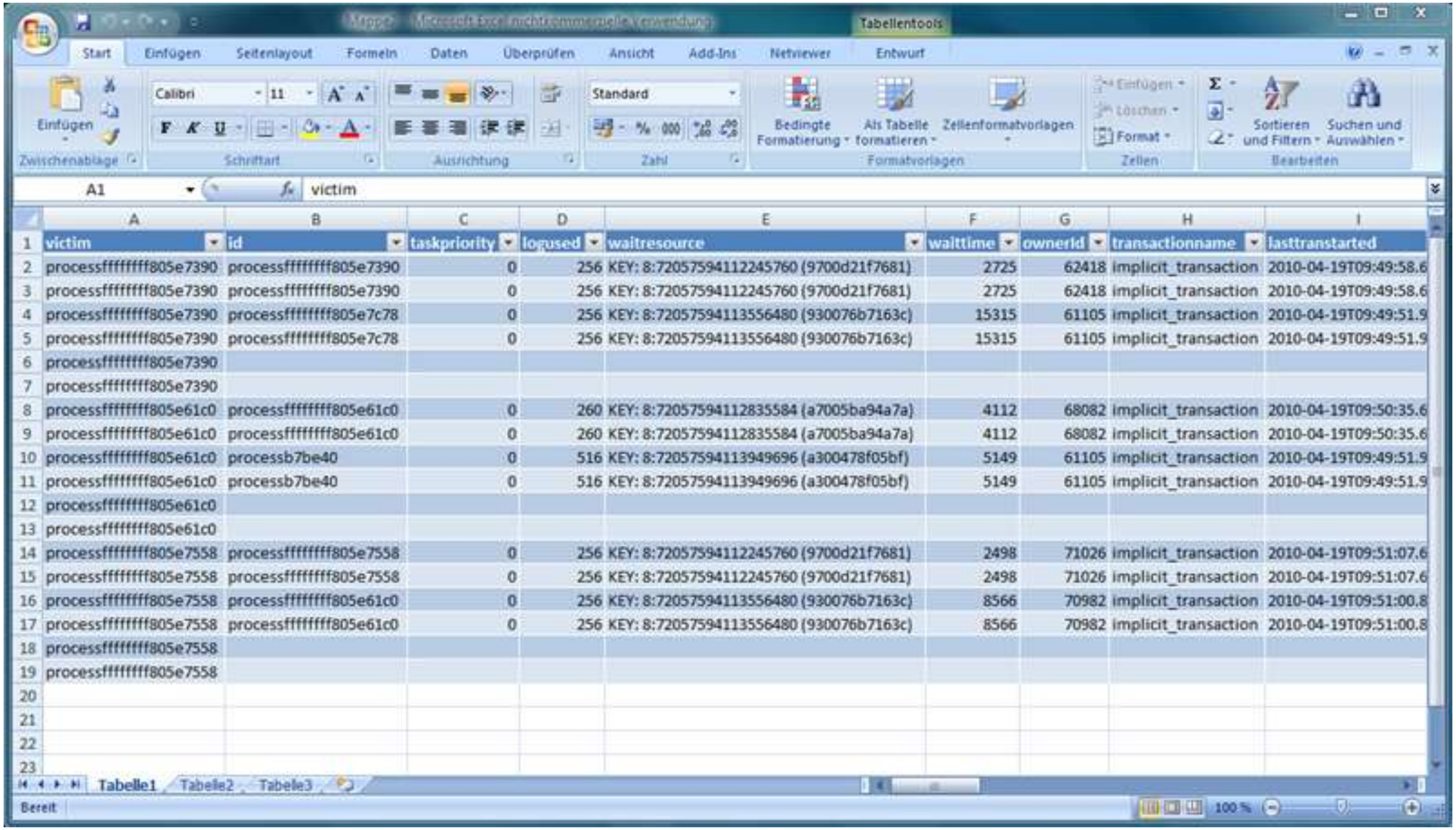
This script has been copied into a SQL Agent **Job** "*SSI: Deadlock Trace*" which starts up automatically whenever the Agent service is started, so this job creates a **persistent background trace** "listening" to Deadlocks.

(some more convenient file management was added)

Hence, a fully automatic and event-triggered deadlock detection process is established!

To investigate the Deadlocks the trace has to be stopped, then the "*Deadlock Graphs*" could be **extracted** into XDL file(s).
The XDL files contain all relevant information about the deadlock – *who is blocking whom, where, when and how* – as an XML structure, hence the data could be **analyzed** with <u>any</u> utility which is capable to deal with XML (e.g. MS Excel, etc.).

Step 4: Deadlock **Analysis**



	victim	id	taskpriority	logused	waitresource	waittime	ownerId	transactionname	lasttranstarted
2	processffffffff805e7390	processffffffff805e7390	0	256	KEY: 8:72057594112245760 (9700d21f7681)	2725	62418	implicit_transaction	2010-04-19T09:49:58.6
3	processffffffff805e7390	processffffffff805e7390	0	256	KEY: 8:72057594112245760 (9700d21f7681)	2725	62418	implicit_transaction	2010-04-19T09:49:58.6
4	processffffffff805e7390	processffffffff805e7c78	0	256	KEY: 8:72057594113556480 (930076b7163c)	15315	61105	implicit_transaction	2010-04-19T09:49:51.9
5	processffffffff805e7390	processffffffff805e7c78	0	256	KEY: 8:72057594113556480 (930076b7163c)	15315	61105	implicit_transaction	2010-04-19T09:49:51.9
6	processffffffff805e7390								
7	processffffffff805e7390								
8	processffffffff805e61c0	processffffffff805e61c0	0	260	KEY: 8:72057594112835584 (a7005ba94a7a)	4112	68082	implicit_transaction	2010-04-19T09:50:35.6
9	processffffffff805e61c0	processffffffff805e61c0	0	260	KEY: 8:72057594112835584 (a7005ba94a7a)	4112	68082	implicit_transaction	2010-04-19T09:50:35.6
10	processffffffff805e61c0	processb7be40	0	516	KEY: 8:72057594113949696 (a300478f05bf)	5149	61105	implicit_transaction	2010-04-19T09:49:51.9
11	processffffffff805e61c0	processb7be40	0	516	KEY: 8:72057594113949696 (a300478f05bf)	5149	61105	implicit_transaction	2010-04-19T09:49:51.9
12	processffffffff805e61c0								
13	processffffffff805e61c0								
14	processffffffff805e7558	processffffffff805e7558	0	256	KEY: 8:72057594112245760 (9700d21f7681)	2498	71026	implicit_transaction	2010-04-19T09:51:07.6
15	processffffffff805e7558	processffffffff805e7558	0	256	KEY: 8:72057594112245760 (9700d21f7681)	2498	71026	implicit_transaction	2010-04-19T09:51:07.6
16	processffffffff805e7558	processffffffff805e61c0	0	256	KEY: 8:72057594113556480 (930076b7163c)	8566	70982	implicit_transaction	2010-04-19T09:51:00.8
17	processffffffff805e7558	processffffffff805e61c0	0	256	KEY: 8:72057594113556480 (930076b7163c)	8566	70982	implicit_transaction	2010-04-19T09:51:00.8
18	processffffffff805e7558								
19	processffffffff805e7558								

Summary

With 100% **on-board features** of SQL Server at **zero costs**, by simply using little TSQL programming, some completely **automatic** and **convenient** processes to detect and investigate blocking and deadlocking situations could be **easily** established!

The recorded data could be used as **an ideal starting point** for further investigation of NAV's C/AL code and the business processes; thus Dynamics NAV partners and customers could proceed the **troubleshooting** to find and implement **appropriate solutions**!

Questions & Answers

Thanks!

Jörg A. Stryk

STRYK System Improvement

http://www.stryk.info

contact@stryk.info