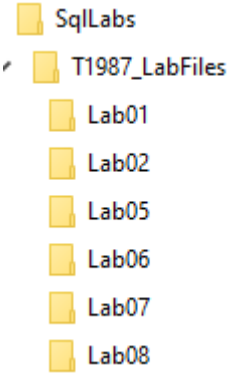
Lab Manual

T1987, Performance Tuning and Optimizing SQL Databases

Lab environment overview

- Your machine name is ***NORTH***.
- If not already done for you, log in to Windows using
  - ***Student***
  - ***myS3cret***
- Copy the lab files to your virtual machine. You find them here: https://karaszi.com/training
  - Use a web browser in the above virtual machine
  - If Virsoft is used: to paste text from the host into the virtual machine, position the cursor, right-click the notepad at top left and select "paste…"
- Extract the files, so you in the root of your C: drive end up with a folder structure looking like below



- You probably want to use ***SQL Server Management Studio (SSMS)*** to do the labs. You are welcome to use Azure Data Studio (ADS) as well, however, the lab instructions assume SSMS.
- You will use the following SQL Server instances:
  - ***Default*** (connect using just the machine name ***NORTH***). This is the main instance for your labs).
  - Some labs might use the ***A*** instance (connect using ***NORTH\A***). The lab instructions will specify if that is the case.
- Login to your SQL Server instance using Windows authentication, unless otherwise specified.
  - (There is a Windows login in your SQL Server for your Windows account, which is a sysadmin.)

- You can revert/reset your databases to "default".
  - There are three bat files in the C:\SqlLabs folder that does this (RESTORE), one for each database.
    - You might need to download these files first, from https://files.karaszi.com/rot/courses/RestoreLabDatabases.zip
  - Note: **Runs as Administrator**
- The lab answers are *not* designed to be used independently. Use the lab instructions, and check the answers when you get stuck, etc.

Lab 1: Architecture, scheduling, and wait statistics

NOTE: Before doing the lab, run below bat file (**double-click** or r**ight-click, Open**).

C:\SqlLabs\T1987_LabFiles\Lab01**Setup.bat**

This will take a couple of minutes, wait until you see "Press any key…" in the cmd window.

(Do not peek at what the files does. You don't want to spoil the fun, do you?)

Ex 1: Document the hardware configuration

You encounter a new SQL Server and you want to document the hardware related configuration of this server. Use whatever means that suits you (queries, GUI in SSMS, Glenn Berry's script). We suggest you use Glenn's script; you will probably find that with one script file you can find out a lot of information about a SQL Server instance.

1. How many CPU cores?
2. What is the *effective* "max worker threads" count?
3. How much memory in the machine? (FYI, dynamic memory might be used in the lab environments hypervisor.)
4. What version and CU level do you have?
    a. Is this a recent level?
5. What edition do you have?
    a. Is Enterprise Edition functionality available for you?

Ex 2: Determine the major wait statistics

Determine what is the highest wait stats on the instance, since it was re-started. Feel free to read up more on this wait stats, if you have the time.

Lab 1 answer suggestions

There are no answer suggestions for this lab. Use the lab instructions.

Lab 2: SQL Server I/O

Ex 1: Determine if a query is I/O bound

Open below file and determine whether the query is I/O bound

C:\SqlLabs\T1987_LabFiles\Lab02**Slow query.sql**

Ex 2: If time permits, improve the performance of the query

Feel free to do this if you have the time. We haven't reached the modules with the necessary information in the course material yet, so don't feel bad if you need to look at the lab answer suggestion.

Can you think of something that might improve the performance?

Feel free to implement that solution and test it. Make sure to clean up after yourself!

Lab 2 answer suggestions

Ex 1: Determine if a query is I/O bound

Alternative 1:

Execute the query and include the actual execution plan.

Open the Properties in SSMS (F4). In the execution plan, click on some *other* operator than the SELECT and *then* click on the SELECT operator. (This is a quirk in SSMS, it looks like the SELECT operator is chosen, but you have to choose some other operator and then the SELECT operator again to actually make it chosen.)

In the properties window, find the WaitStats section and see which is at the top.

Alternative 2:

Execute below query, from the same window as where you executed your slow SELECT. Or if you do it from a different window, change the session_id to match the right one.

```
SELECT * FROM sys.dm_exec_session_wait_stats
WHERE session_id = @@SPID
ORDER BY wait_time_ms DESC
```

Ex 2: If time permits, improve the performance of the query

One way to improve performance is to have a non-clustered index on the UnitPrice column. Note that indexes are also overhead, so which indexes you want on a table should be chosen with care.

(There are other options as well (indexed view, Columnstore index, for instance), but this is perhaps the most obvious option.)

Create below index, test the query again and then drop the index.

```
CREATE INDEX UnitPrice ON FactResellerSalesXL(UnitPrice)
GO

--Re-test your query. Empty the data cache before executing the query.
```

Note that we might still have the same wait stats at the top. There is always something at the top. But we have a lot less of that wait stats, and the query is a lot quicker.

```
--Clean up
DROP INDEX UnitPrice ON FactResellerSalesXL
GO
```

Lab 3: Database structures

Ex 1: Verifying Instant File Initialization

Check if instant file initialization is on.

If it isn't on, then turn it on.

Ex 2: Verifying number of tempdb datafiles

Check how many CPU cores that your instance is seeing.

Check above against the number of tempdb files.

Let us assume that it doesn't match (even if it does). Add 2 files. (Say that you for instance have 2 files and the number of CPU cores is 4. The details aren't important, this is only an exercise.) You want to have the same startup-size and autogrow for all files. Use for instance https://sqlblog.karaszi.com/managing-tempdb/ for a query to check the startup size for your current tempdb files.

(It doesn't matter if you after the lab have more files than cores.)

After adding the files:

Check the current state for your temp files, re-start SQL Server and check again.

Lab 3 answer suggestions

Ex 1: Verifying Instant File Initialization

```
--Use DMV to check
SELECT servicename, instant_file_initialization_enabled
FROM sys.dm_server_services

--Check the errorlog file

--Use undocumented proc to check errorlog file
EXEC sp_readerrorlog 0, 1, 'instant'
```

Turn it on:

- Local Security Policy (secpol)
- Local Policies
- User Right Assignment
- Perform volume maintenance tasks
- Add the name of your service account:
    - NT SERVICE\MSSQLSERVER
- Restart the database engine instance (make sure you re-tart the right instance, MSSQLSERVER)
- Verify that it is on

Ex 2: Verifying number of tempdb datafiles

Check number of CPUs:

SSMS, Object Explorer:

- Right-click the instance, Properties, the General Page
- Or check the Processors page

```
--By checking number of online schedulers
SELECT * FROM sys.dm_os_schedulers WHERE status = 'VISIBLE ONLINE'

--By checking the sys.dm_os_sys_info DMV
SELECT cpu_count FROM sys.dm_os_sys_info
```

Check number of tempdb files

There are several ways to do this. One is to check tempdb-sys-database_files:

```
SELECT * FROM tempdb.sys.database_files WHERE type_desc = 'ROWS'
```

Add a couple of data files to tempdb

Check start-up size for current tempdb files

```
--Start-up size for tempdb
SELECT
DB_NAME(database_id) AS db_name_
,file_id
,name
,physical_name
,size * 8/1024 AS size_MB
,type_desc
,CASE WHEN is_percent_growth = 1 THEN CAST(growth AS varchar(3)) + ' %' ELSE
CAST(growth * 8/1024 AS varchar(10)) + ' MB' END AS growth
,max_size * 8/1024 AS max_size_MB
FROM master.sys.master_files
WHERE DB_NAME(database_id)  = 'tempdb'
ORDER BY db_name_, type, file_id
```

Use SSMS to add the files, or TSQL directly as per below example (which you might have to adjust according to how many files you currently have)

```
ALTER DATABASE [tempdb]
ADD FILE ( NAME = N'temp5', FILENAME =
N'C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\temp3.ndf' ,
SIZE = 8MB , FILEGROWTH = 64MB )
GO
ALTER DATABASE [tempdb]
ADD FILE ( NAME = N'temp6', FILENAME =
N'C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\temp4.ndf' ,
SIZE = 8MB , FILEGROWTH = 64MB )
GO
```

Check the current state for your temp files, re-start SQL Server and check again.

Lab 4: Memory usage

Ex 1: Set max server memory

Set max server memory to a reasonable value. Determine how much memory you have in the machine, and then use one for the formulas that exists "out there".

Note that depending on the machine setup, whether the hypervisor is using dynamic memory etc, you might see a strange/very low value. This is something that you want to watch out for in real life. If that is the case, for the sake of the lab, let us assume that you have 32 GB memory in the machine.

Ex 2: Work with query that has "memory issues"

You have a query which you suspect is using a lot of memory. You want to verify that.

Run below Setup section once, and then the actual query section as many times as you need. When you're done, run the Clean-up section.

How can you determine that this query uses a lot of memory?

Can you think of a way to reduce the memory usage? (FYI, there are things you can do that we haven't talked about yet.)

```
--Setup
USE AdventureworksDW
ALTER DATABASE SCOPED CONFIGURATION SET ROW_MODE_MEMORY_GRANT_FEEDBACK = OFF
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_MEMORY_GRANT_FEEDBACK = OFF
DBCC FREEPROCCACHE
GO

--The actual query
DECLARE @dd1 datetime = '20150101', @dd2 datetime = '20150102'
SELECT DISTINCT f.CarrierTrackingNumber, f.CustomerPONumber, CAST(f.CurrencyKey AS
varchar(max))
FROM FactResellerSalesXL AS f
WHERE DueDate > = @dd1 AND DueDate < @dd2
ORDER BY CarrierTrackingNumber, CustomerPONumber
OPTION(MAXDOP 1)
GO

--Clean-up
ALTER DATABASE SCOPED CONFIGURATION SET ROW_MODE_MEMORY_GRANT_FEEDBACK = ON
ALTER DATABASE SCOPED CONFIGURATION SET BATCH_MODE_MEMORY_GRANT_FEEDBACK = ON
```

Lab 4 answer suggestions

Ex 1: Set max server memory

There are several ways to determine the amount of memory in a machine. Below is one:

```
SELECT total_physical_memory_kb/1024 AS PhysicalMemoryMB
FROM sys.dm_os_sys_memory
```

Let us assume (regardless of what above show us) that we have 32 GB.

Use any of these recommendations:

- https://www.brentozar.com/blitz/max-memory/
- https://sqlblog.karaszi.com/setting-max-server-memory/

Set it to approx. 26 GB, turn on show advanced options first:

```
EXEC sp_configure 'show advanced options', 1 RECONFIGURE
```

```
EXEC sp_configure 'max server memory', 26357 RECONFIGURE
```

Ex 2: Work with query that has "memory issues"

One way to determine high memory usage is looking at the actual execution plan. You will here also see if SQL Server allocates a lot of memory, but it turns out that it didn't need all that memory. Or the other way around, it needed more memory than allocated, so the query spilled to disk.

Another way is to capture events using an Extended Event trace, which will be discussed in a later module.

If you're a fan of the sp_blitz procedures, then you might want to use below which uses the plan cache:

```
EXEC sp_BlitzCache @SortOrder = 'memory grant';
```

As for reducing memory requirements, things that can be done include:

- Get rid of that ridiculous CAST in the SELECT list. SQL Server assumes approx. 4 KB for each value, which then will be used for sort operation (for all of the rows that SQL Server *assumes* will be returned).
- Change from variables to literals in the WHERE clause. The optimizer has no knowledge of the variable value, and uses hard-wired constants when it estimates number of rows.
- Add the RECOMPILE option, but you now have to pay for plan compilation for each execution.

Lab 5: Concurrency

Ex 1: Improve concurrency

You have a situation where users are complaining about bad performance. You suspect that you have concurrency issues, i.e., blocking. Below is the workload for this lab, i.e., this represents the workload on your SQL Server:

Close all query windows in SSMS. Open below two files in SSMS, so you now have (only) two query windows:

- C:\SqlLabs\T1987_LabFiles\Lab05\DoUpdates.sql
- C:\SqlLabs\T1987_LabFiles\Lab05\DoSelects.sql

Optional: Put them side-by-side in SSMS (alt-W, V if you prefer keyboard shortcut).

- Window menu
- New Vertical Tab Group

Execute both query windows simultaneous. It doesn't matter which you start first, but start the second pretty directly after you started the first one. They will run for about 1 minute. Switch to the "messages" tab in each query window to get some text feedback while the script is running.

Take a note of how many SELECTs you managed to do in one minute.

You suspect that you have blocking issues. *How can you verify that?* (Feel free to check the answer suggestions if it isn't obvious…)

Set the read_committed_snapshot database setting to 1. You cannot have any open query windows while you change this setting, the ALTER command will be blocked. If you use the GUI in SSMS, it will close the connections for you.

Re-run the tests. Did you get an increased number of SELECTs?

When you're done, reset the versioning setting for the database back to default!

Ex 2: Capture deadlock information into a trace

You suspect that you have deadlocks in your system. You want to catch them in a trace so you can send that trace files to the developers.

Create the trace to catch the deadlocks. You want the XML based deadlock info, so that the developers have details about the deadlock situation. Have the trace go to a file.

We haven't discussed Extended Events yes. Feel free to try on your own, use the GUI in SSMS etc. You can of course find such a trace definition in the answer suggestions below.

Use below two files to generate the deadlock.

- C:\SqlLabs\T1987_LabFiles\Lab05\Deadlock1.sql
- C:\SqlLabs\T1987_LabFiles\Lab05\Deadlock2.sql

Use the comments in the file to see in what order you should execute the commands. Start with the Deadlock1 file.

Note: If you don't get a blocking and a subsequent deadlock, you still have read_commited_snapshot turned on for the database, from previous exercise. Turn it off and re-try.

Check your deadlock trace. For instance in SSMS, Management, Extended Events, Sessions, <name of your trace>, right-click the target (might be package0.event_file), View target data.

Lab 5 answer suggestions

Ex 1: Improve concurrency

Follow the lab instructions.

How can you determine if you have lots of blocking?
Check out the wait stats. You can use Glenn Berry's query for this (search in his file for "dm_os_wait", at the time of writing this lab it is query number 39).

To change the database setting to get versioning instead of blocking:

```
USE master
GO
ALTER DATABASE Adventureworks SET SINGLE_USER WITH ROLLBACK IMMEDIATE
WAITFOR DELAY '00:00:02'
ALTER DATABASE Adventureworks SET READ_COMMITTED_SNAPSHOT ON
WAITFOR DELAY '00:00:02'
ALTER DATABASE Adventureworks SET MULTI_USER WITH ROLLBACK IMMEDIATE
GO
```

When you're done, reset the versioning setting for the database back to default:

```
USE master
GO
ALTER DATABASE Adventureworks SET SINGLE_USER WITH ROLLBACK IMMEDIATE
WAITFOR DELAY '00:00:02'
ALTER DATABASE Adventureworks SET READ_COMMITTED_SNAPSHOT OFF
WAITFOR DELAY '00:00:02'
ALTER DATABASE Adventureworks SET MULTI_USER WITH ROLLBACK IMMEDIATE
GO
```

Ex 2: Capture deadlock information into a trace

Below is what a deadlock trace configuration can look like:

```
CREATE EVENT SESSION [Deadlocks] ON SERVER
ADD EVENT sqlserver.xml_deadlock_report(
    ACTION(sqlserver.database_name))
ADD TARGET package0.event_file(SET
filename=N'C:\DbFiles\Deadlocks',max_file_size=(1024),max_rollover_files=(3))
WITH (MAX_MEMORY=4096
KB,EVENT_RETENTION_MODE=ALLOW_SINGLE_EVENT_LOSS,MAX_DISPATCH_LATENCY=30
SECONDS,MAX_EVENT_SIZE=0
KB,MEMORY_PARTITION_MODE=NONE,TRACK_CAUSALITY=OFF,STARTUP_STATE=OFF)
GO

--Here's how to autostart when SQL Server starts
-- ALTER EVENT SESSION [Deadlocks] ON SERVER
--  WITH (STARTUP_STATE=ON)

--Start the trace
ALTER EVENT SESSION Deadlocks
ON SERVER
STATE = START
```

Lab 6: Statistics and indexes

NOTE: Before doing the lab, run below bat file (**double-click** or r**ight-click, Open**).

C:\SqlLabs\T1987_LabFiles\Lab06**Setup.bat**

Do not peek at what the files does. You don't want to spoil the fun, do you?

NOTE: Do this lab in the AdventureworksDW database.

Ex 1: Improve a query, the easy way

You have below query which you wish could execute a lot quicker. It currently takes some 30-60 seconds. Adding indexes or changing the query is not an option. You wish you could find that "turbo-button" – just one configuration setting. Find it. Press the "turbo-button". Re-run the query and see if it is quicker.

- C:\SqlLabs\T1987_LabFiles\Lab06\SlowQuery1.sql

Optional: exactly *why* is the query quicker now?

Ex 2: Improve a query, the traditional way

You have a query which uses more resources than you find reasonable. Check the number of logical I/O it does. Improve it and verify.

- C:\SqlLabs\T1987_LabFiles\Lab06\SlowQuery2.sql

Ex 3: Improve a query, by query re-write

You have a query which is slow. You have a feeling that the developer isn't top notch when it comes to writing performant SQL. Improve the query.

- C:\SqlLabs\T1987_LabFiles\Lab06\SlowQuery3.sql

Lab 6 answer suggestions

Ex 1: Improve a query, the easy way

In this case, we can up the compatibility level so we get batch mode on rowstore:

```
ALTER DATABASE current SET COMPATIBILITY_LEVEL = 150 --2019
```

Ex 2: Improve a query, the traditional way

One way to see number of I/O operations is to turn on below and then check the "messages" tab after you executed a query:

```
SET STATISTICS IO ON
```

Just create an index on the OrderDate column, the query is pretty selective so no need to mess about with included columns.

```
CREATE INDEX ix_OrderDate ON FactResellerSalesXL(OrderDate)
```

Ex 3: Improve a query, by query re-write

Below are two query re-writes that will allow SQL Server to seek through the index on the OrderDate column:

```
SELECT *
FROM FactResellerSalesXL AS s
WHERE CAST(OrderDate AS date) = '2014-04-02'

SELECT *
FROM FactResellerSalesXL AS s
WHERE s.OrderDate >= '2014-04-02' AND s.OrderDate < '2014-04-03'
```

Lab 7: Execution plans

NOTE: Before doing the lab, run below bat file (**double-click** or r**ight-click, Open**).

C:\SqlLabs\T1987_LabFiles\Lab07**Setup.bat**

Do not peek at what the files does. You don't want to spoil the fun, do you?

Ex 1: Improve performance of the GetOrderDetailsReseller stored procedure

Execute the procedure as given below. The procedure is quick, but it is executed *very frequently* and the developer ask you to verify that it doesn't use excessive resources. Look at the execution plan and output from SET STATISTICS IO ON.

```
EXEC GetOrderDetailsReseller @ResellerKey = 509
```

How many logical reads does it do? Try to get it to somewhere around 100,000 (logical) read operations. Make that happen and verify that you get the expected execution plan.

You now get the message that this still uses way too much resources and that the CPU seems to be under high load. We want to get logical reads in the hundreds. Make it so.

Ex 2: Improve performance of the GetOrderDetailsDueDate stored procedure

Execute the procedure as given below. Do not read the source code for the procedure initially, only look at the actual execution plan. Carefully read the Warnings for the SELECT operator.

```
EXEC GetOrderDetailsDueDate @DueDateKey = '20050304'
```

Try to improve the procedure by adding index only, inspired by Exercise 1 above. Verify the result using execution plan and statistics io

If above didn't make a huge difference, do a query re-write to make it use less resources.

Lab 7 answer suggestions

Ex 1: Improve performance of the GetOrderDetailsReseller stored procedure

```
--First attempt, a non-covering index on ResellerKey
DROP INDEX IF EXISTS ResellerKey ON FactResellerSalesXL

CREATE INDEX ResellerKey ON FactResellerSalesXL(ResellerKey)


--Second attempt, make it a covering index
DROP INDEX IF EXISTS ResellerKey ON FactResellerSalesXL

CREATE INDEX ResellerKey ON FactResellerSalesXL(ResellerKey)
INCLUDE (ProductKey, EmployeeKey, SalesAmount )
```

Ex 2: Improve performance of the GetOrderDetailsDueDate stored procedure

```
--Remove index if exists
DROP INDEX IF EXISTS DueDateKey ON FactResellerSalesXL

--Add index. It is your choice if you want to make it covering.
CREATE INDEX DueDateKey ON FactResellerSalesXL(DueDateKey)
INCLUDE (ProductKey, EmployeeKey, SalesAmount )
GO

--Query re-write, to make it SARGable.
CREATE OR ALTER PROC GetOrderDetailsDueDate
@DueDateKey int
AS
SELECT f.ProductKey, f.ProductKey, f.EmployeeKey, f.SalesAmount
FROM FactResellerSalesXL AS f
WHERE f.DueDateKey = @DueDateKey
GO
```

Lab 8: Plan caching and recompile

Ex 1: Fixing stored procedure with plan issues

NOTE: Before doing this exercise, run below bat file (**double-click** or r**ight-click, Open**).

C:\SqlLabs\T1987_LabFiles\Lab08**Setup_1.bat**

Above will take about a minute to execute.

The GetOrderDetails procedure in the AdventureworksDW database procedure sometimes execute very quickly, but sometimes it takes several seconds. This procedure need to go sub-second. You have been called in because the procedure is slow now:

```
EXEC GetOrderDetails @OrderDateKey = 20070520, @DiscountAmount = 0
```

Execute above, check execution plan and statistics io. Determine why it is slow and come up with possible ways to handle the situation.

Do ***NOT*** empty the plan cache. If you do, re-run the setup script.

Ex 2: Server is gradually slowing down

NOTE: Before doing this exercise, run below bat file (**double-click** or r**ight-click, Open**).

C:\SqlLabs\T1987_LabFiles\Lab08**Setup_2.bat**

Above will take about a minute to execute.

Your users claim that the application becomes slower and slower, and you have made the connection that a re-start of SQL Server seem to "fix" this.

You expect poor plan re-use. I.e., the plan cache gets polluted with plans that aren't re-used. Verify your assumption.

How can you fix this?

Lab 8 answer suggestions

Ex 1: Fixing stored procedure with plan issues

The problem is parameter sniffing. When you had run the lab setup script, you have a plan in cache that assumes about 650 rows for the predicate on the DiscountAmount column. Where with our parameters, there are in fact about 5,300,000 rows. There are several ways to handle the situation.

- Create the proc using WITH RECOMPILE
  - You pay for optimization for each execution and for each query in the proc
- Add OPTION(RECOMPILE) for the problematic SELECT statement only
  - You pay for optimization for each execution, but only for the problematic query
- Use OPTION(OPTIMIZE FOR) for the parameter in question and specify something that gives a good plan for the typical execution
  - You don't pay for optimization for each execution, but you can get hurt if you have executions with parameter values that would benefit greatly from a different plan
- Use Query Store and force the "good plan"

Below is one of the above possible ways to handle this:

```
CREATE OR ALTER PROC GetOrderDetails
@OrderDateKey int, @DiscountAmount float
AS
SELECT f.ProductKey, f.ProductKey, f.EmployeeKey, f.SalesAmount
FROM FactResellerSalesXL AS f
WHERE f.OrderDateKey = @OrderDateKey
AND DiscountAmount = @DiscountAmount
OPTION(RECOMPILE)
GO
```

Ex 2: Server is gradually slowing down

You can check sys.dm_exec_cached_plans, and see how many times each plan has been executed. Here are a couple of example queries:

```
--Overall re-use
SELECT
 c.usecounts
,COUNT(*) AS antal
,SUM(c.size_in_bytes / (1024 * 1024.0)) AS size_in_MB
FROM sys.dm_exec_cached_plans AS c
GROUP BY c.usecounts
ORDER BY usecounts ASC

--Per database, only those with 1 re-use
SELECT
 c.usecounts
,DB_NAME(CAST(a.vålue AS sysname))
,COUNT(*) AS antal
,SUM(c.size_in_bytes / (1024 * 1024.0)) AS size_in_MB
FROM sys.dm_exec_cached_plans AS c
 CROSS APPLY sys.dm_exec_plan_attributes(c.plan_handle) AS a
WHERE a.attribute = 'dbid'
AND c.usecounts = 1
GROUP BY c.usecounts, DB_NAME(CAST(a.value AS sysname))
```

1. Talking to the developers and have them parameterize their SQL is likely the best option.
2. Setting optimize for ad-hoc workloads at database or server level will stop SQL Server cache those queries and save memory.
3. Another option can be the PARAMETERIATION FORCED database setting.

If you have the time, feel free to try 2 and/or 3 above.

Lab 9: Extended Events

Ex 1: Capturing SQL commands from an application

You want to know what SQL is submitted by a client application when a certain action is taken in the application. For the purpose of this lab, we will use Activity Monitor in SSMS as the client application. You can use the "New Session Wizard" or "New Session…" to configure the trace, as you feel comfortable with.

- Create a new trace.
- Do not use a template.
- Add the sql_statement_starting event
- Configure a file target, for instance to the C:\DemoDatabases folder
- Start the trace
- Hook up the "Watch Live Data" to the trace window
- Open Activity Monitor in SSMS (right-click the instance in Object Explorer)
- Return to the Live Window
- Stop the data Feed
- Scroll through the SQL commands
- Expand your trace and open the event file
- Stop the trace

Ex 2: Modify the trace definition

You realize that above wasn't really what you wanted. You want to see the cost of the events, and also see the database name, so you can group by it. You want to keep trace you defined above, so you want to "copy" it and modify that "copy".

- Script the trace definition (as Create) you created in Ex 1
- Modify the name of the trace and also the file name
- Execute the CREATE command
- Modify the trace you just created
- Remove the sql_statement_starting event
- Add the sql_statement_completed event
- Add the action database_name to this event
- Save the trace, start it, let Activity Monitor run for a minute or two with the trace started
- Stop the trace.
- For the file target, do View Target Data
- Select any event
- Add the database_name field as a column to above table
- Also add the duration and row_count fields as columns
- Group by the database_name
- Sum over duration
- You can now see the sum of duration for each database represented in the trace
- Stop the trace, if it isn't stopped already

Ex 3: Capturing strange events

This is a more informal exercise, do it if you feel like it and use your imagination.

- Create the “Looking for Strange” trace that you find below: https://karaszi.com/looking-for-strange
- Start the trace
- Let it run for a while
    - Have Activity Monitor open
    - Play around in Objects explorer
    - For example:
        - Modify an Agent job
        - Open Design for a table in some database
        - Create a database diagrams
- When you are happy with generated activity on your SQL Server:
- View Target Data for the event_counter target
- Did you get any events? Anything that means something to you?

Lab 9 answer suggestions

Ex 1: Capturing SQL commands from an application

```
--Create a new trace
CREATE EVENT SESSION [Capture SQL starting events] ON SERVER
ADD EVENT sqlserver.sql_statement_starting
ADD TARGET package0.event_file
(SET filename=N'C:\DemoDatabases\Capture SQL starting events')
GO

--Start the trace
ALTER EVENT SESSION [Capture SQL starting events] ON SERVER
STATE = START
```

- Hook up the “Watch Live Data” to the trace window
- Open Activity Monitor in SSMS (right-click the instance in Object Explorer)
- Return to the Live Window
- Stop the data Feed
- Scroll through the SQL commands
- Expand your trace and open the event file

```
--Stop the trace
ALTER EVENT SESSION [Capture SQL starting events] ON SERVER
STATE = STOP
```

Ex 2: Modify the trace definition

```
--Create the new (modified) trace definition
CREATE EVENT SESSION [Capture SQL completed events] ON SERVER
ADD EVENT sqlserver.sp_statement_completed(
    ACTION(sqlserver.database_name))
ADD TARGET package0.event_file
(SET filename=N'C:\DemoDatabases\Capture SQL completed events')
GO

--Start the trace
ALTER EVENT SESSION [Capture SQL completed events] ON SERVER
STATE = START
```

- Let Activity Monitor run for a while with the trace started
- Stop the trace.
- For the file target, do View Target Data
- Select any event
- Add the database_name field as a column to above table
- Also add the duration and row_count fields as columns
- Group by the database_name
- Sum over duration
- You can now see the sum of duration for each database represented in the trace

```
--Stop the trace
ALTER EVENT SESSION [Capture SQL completed events] ON SERVER
STATE = STOP
```

Ex 3: Capturing strange events

There are no answer suggestions for this exercise. Use the lab instructions.