IDERA® | SQL Server Whitepaper

DEMYSTIFYING DEBUGGING TECHNIQUES WITH SQL SERVER

BY PINAL DAVE

INTRODUCTION

The greatest happiness for a father is to see our children grow in front of our very own eyes. My daughter has been my source of inspiration for a number of things that I face in life. It is like living your childhood all over again with a twist. I have nothing to complain about but everything to cherish. Recently, I had a unique opportunity to go to my daughter's school for the parent-teachers meeting.

In my last visit to the school, I had an opportunity to observe almost every parent around and how they were handling their kids. There was something I noticed consistently that is worth a mention. The kids surely have a lot of energy and are filled with questions. There wasn't a minute where I would hear a kid ask – "Why, papa?", or "Why is it like this?" This was the consistent questioning to understand more about things happening around them, and they are explorers. The more I watched the whole exercise, the more I started enjoying the day with my daughter. It taught me interesting things that I think are worth sharing. If you ask me in software terms, the kid is debugging each and every result in front of them as they understand the reason.

Just the need to question why things are different and why they are behaving the way they are is a typical instinct of a Developer writing code. Many times I write some pseudo code and when I translate it into actual code, I get baffled with why the result is so different while my logic says something else. This leads me to debug the whole exercise.

As a SQL Server developer, the fundamental fun of debugging started in my early days using the PRINT command inside SQL Server. I personally get a chance to see these debugging practices still alive in lot of production code. In this whitepaper, let me take a chance to introduce you to the debugger techniques introduced with SQL Server 2012. If you are a Visual Studio C# or a .NET developer, these techniques are so similar that you will never forget them. Yes, it is important to know and note that the SQL Server Management Studio IDE is based on the Visual Studio shell. Hence the goodness of VS carries forward with Management Studio too.

SHOW ME THE CODE!!!

The idea of this paper is not to write lengthy code blocks and show you how a debugger might potentially work. We will use a simple looping block as shown below to check how a debugger will work.

The code is simple as it initializes a variable. We have a *WHILE* block which loops while incrementing the variable by 20. In every 200 values achieved, we are printing the same as part of the code. The code is proposed to run till 10000.

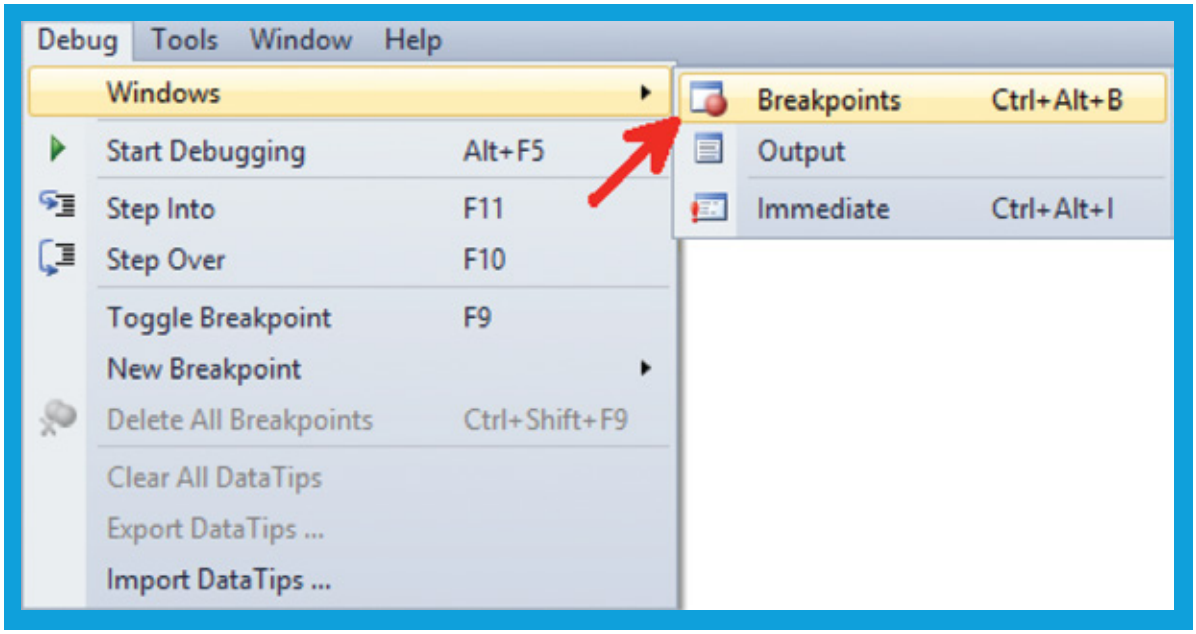
```
-- Simple loop for Debugger Basics
DECLARE @loop INT = 0
WHILE (@loop < 10000)
BEGIN
   IF (@loop%200 = 0)
      -- Used to track our values
   PRINT @loop

   SET @loop += 20
END
```

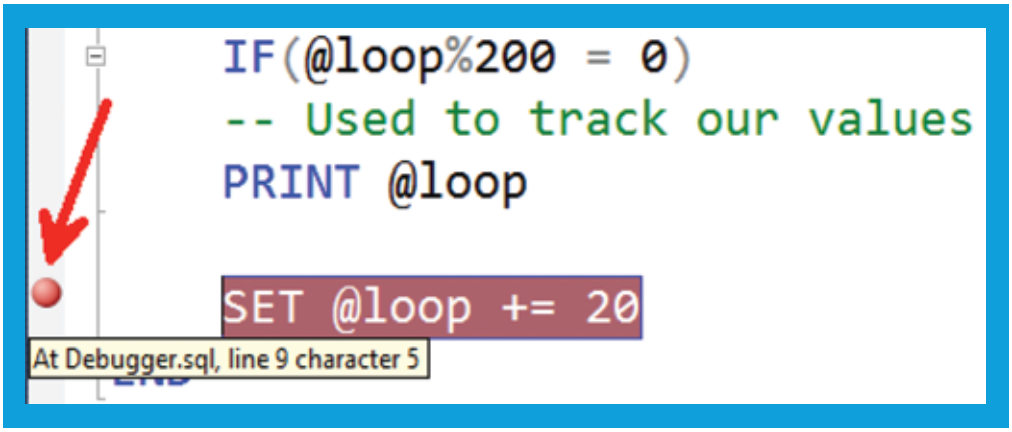
There is absolutely no rocket science to this code. Let us assume this can be code that runs in your production environment where we need to debug how the variable value increases in each of the passes.

DEBUGGER – GETTING STARTED

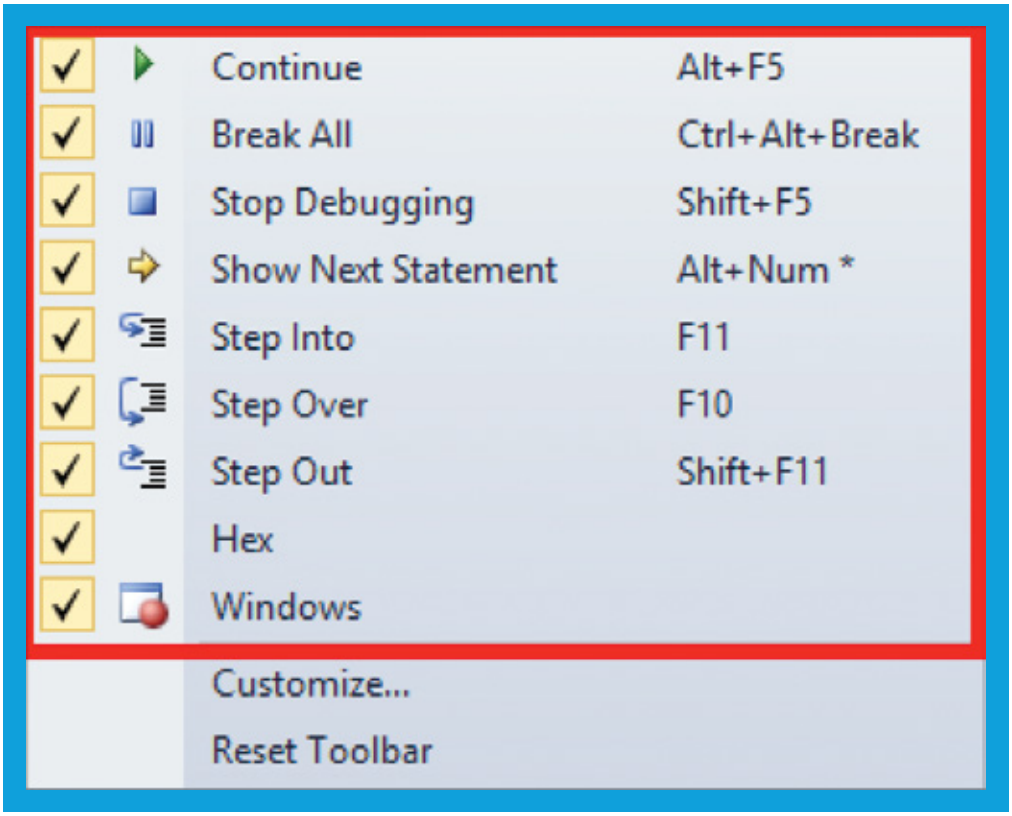
There are a number of ways we can start the whole debugging process. Below shown is our debugger option from the toolbar. Select the **Debug** tab, hover over **Windows** and select **Breakpoints**.



The other way to start the debugger at a specific location is to use the left side ribbon – a “Grey Band” in our SQL Server Management Studio script window. For simplicity reasons, I have shown this in the figure below. So as you can see, enabling the debugger is really easy now.

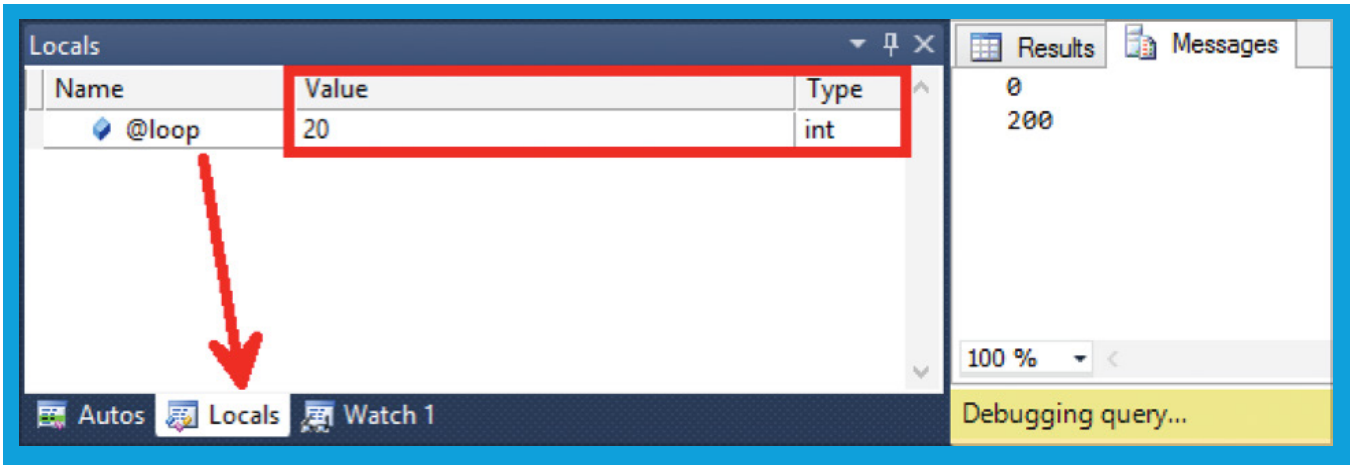


Once the debugger has been enabled and we need to go through the code, we need to know what the other options available for debugging are. The toolbar shows us the list of options we can start using with the debugger. Some of them include **Break All**, **Stop Debugging**, **Next Statement**, **Step Into**, **Step Over** and so on.



WATCHING FOR VALUES

We know the values can be easily found using the *PRINT* statement inside our code block. But once the debugger is started, enable the Locals window. This is the window that allows us to look at all the local variables in the current batch. For our example, the local variable in the batch is @Loop.



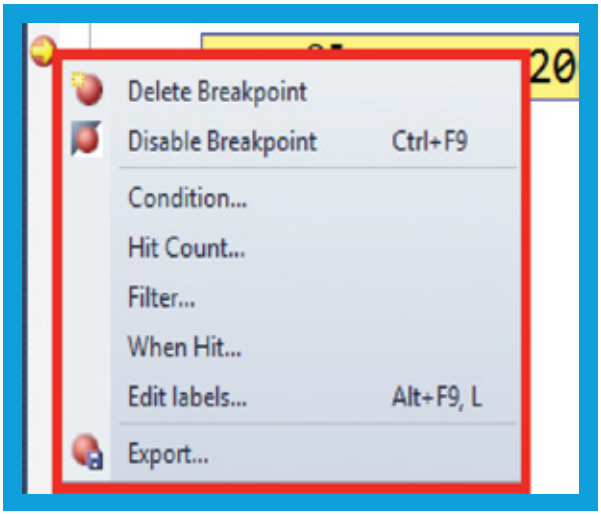
The same value as per the flow of code gets printed in the message window too as shown in the diagram above.

ADVANCED DEBUGGING OPTIONS

Watching for the locals is just one part of the story. As developers there are advanced options that we can start using with SQL Server Management Studio too. If we get an opportunity to right click the Debug point, we will be presented with a list of options that I would like to explain:

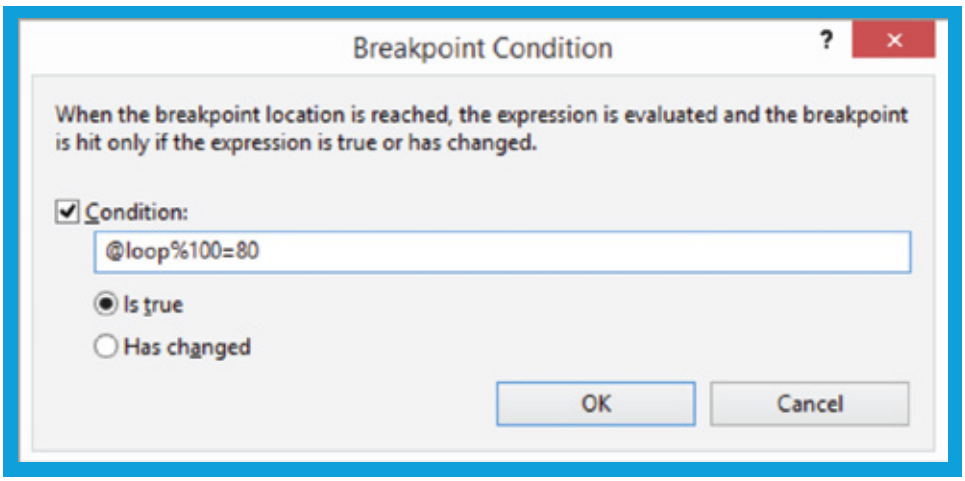
- Condition
- Hit Counter
- Filter
- When Hit

All of these options are handy and can be of great help to developers who are debugging their code. In the next section, we will talk about them quickly to understand how they can be used.



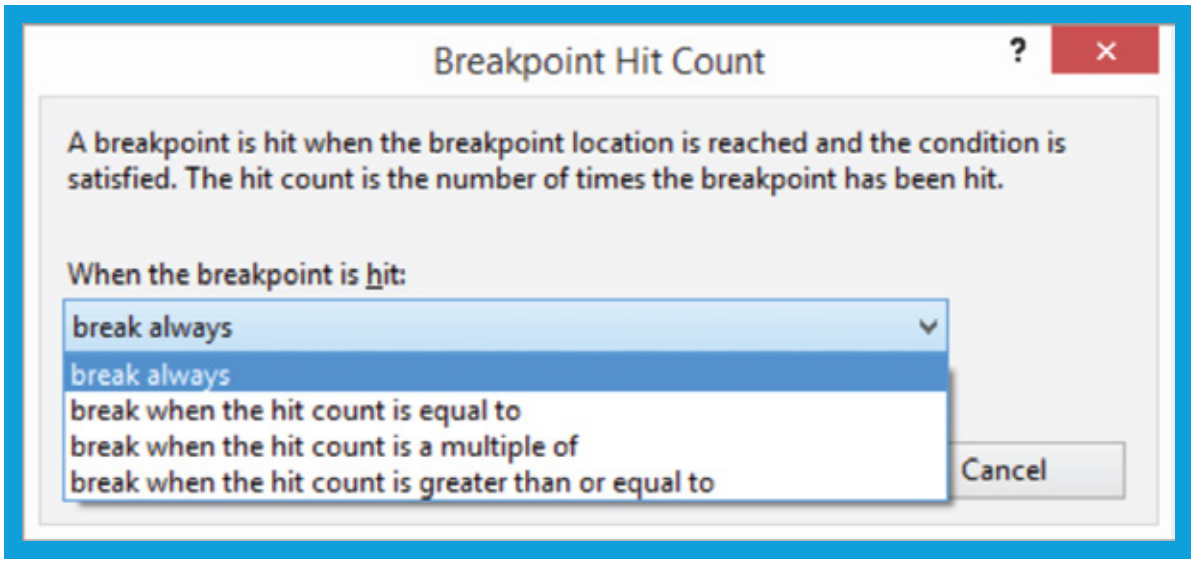
BREAKPOINT CONDITION

As soon as we select this option, we will get a dialog like the one shown below. Here we can add an additional condition which when satisfied the debugger will prompt. In our example we have put an additional condition that the debugger needs to stop when we hit multiples of 80.



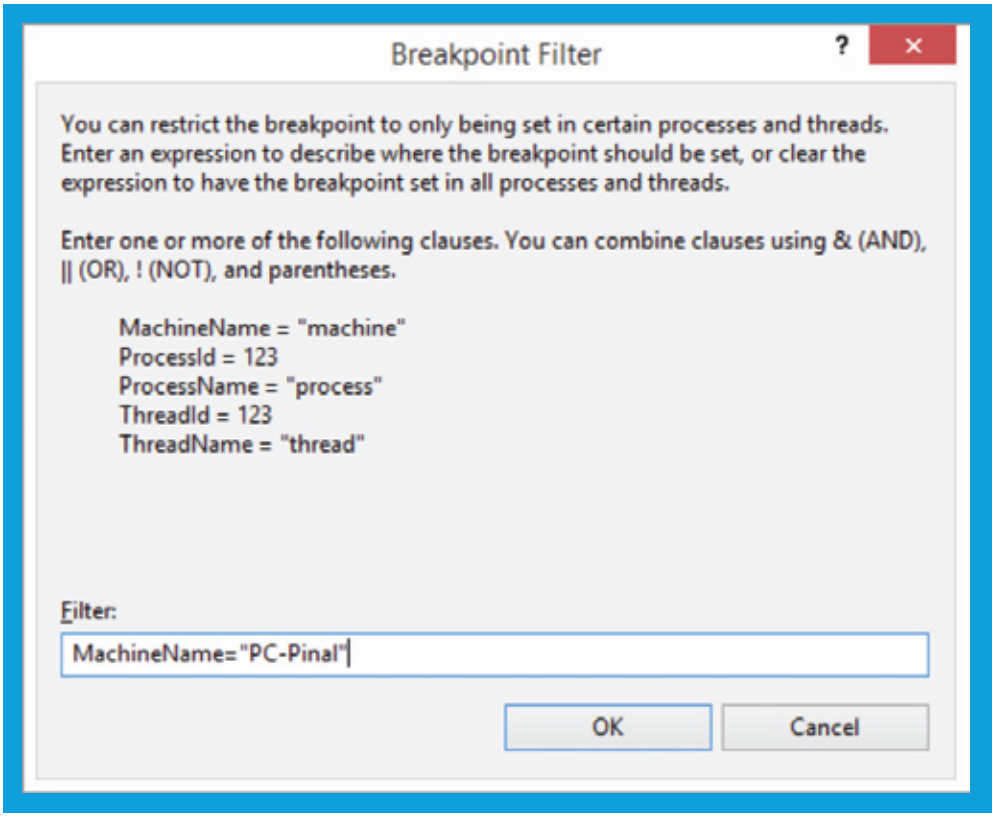
BREAKPOINT HIT COUNT

This option lets us configure what needs to be done when the breakpoint is hit or when the codeflow reaches the breakpoint location. We have a number of options from “break always”, “when the hit count is equal to something”, “when count is a multiple of something” and so on. Use these if you want to skip to specific code flow when the variable hits one of these conditions.



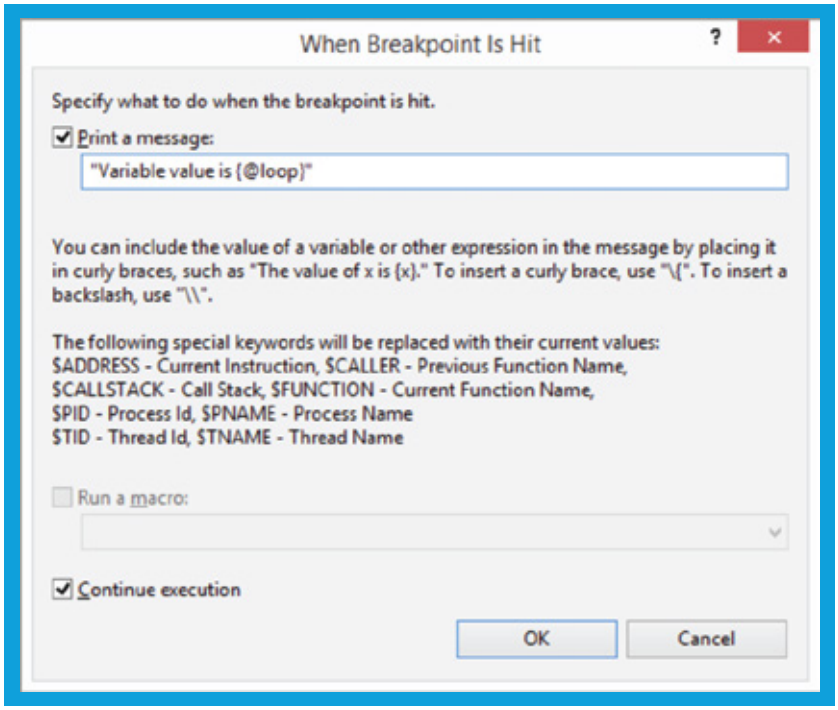
BREAKPOINT FILTER

In this option for breakpoint, we are adding additional filters to the code block. Assume a case when the code is being run in a multi-user scenario. Here we would like to restrict the condition so that the code needs to be debugged only when the machine name is from Pinal. These additional filter blocks can be configured in this “Filter” block. In the below diagram we have added a filter to debug only when the machine is “PC-Pinal”.

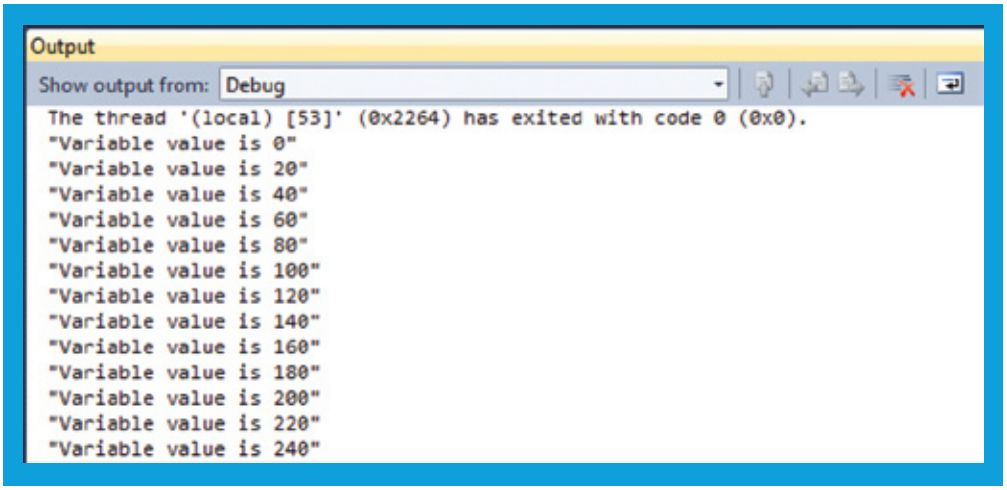


WHEN A BREAKPOINT IS HIT

This is like our very own conventional block of *PRINT* statement that we can also write in the options. It shows what can be done as we hit the breakpoint and how we can visualize the values differently. There are additional functions that can be added as part of our print which can vary from Process ID, Thread ID executing and so on. In our figure below, we have added the value of @Loop to be printed whenever we reach the breakpoint.



A typical output of the breakpoint is hit is shown below. We are printing all the values as we cross the Breakpoint location.



If we keep playing around with what is available with the Visual Studio projects, we can explore more options. As a developer of SQL Server, these are wonderful additions to the existing *PRINT* statements.

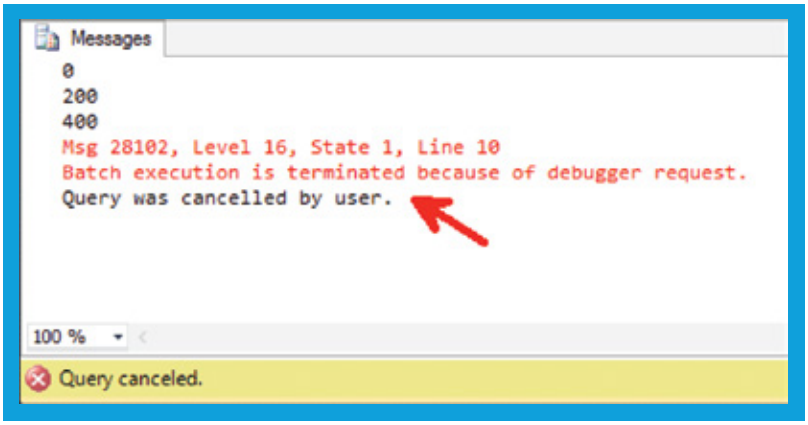
USEFUL DEBUGGER SHORTCUTS

We have shown a number of shortcut keys at various points in this whitepaper. Here is a summary of the shortcut keys.

ACTION	SHORTCUT
Toggle breakpoint	**F9**
Enable breakpoint	**CTRL+F9**
Stop debugging	**SHIFT+F5**
Step into	**F11**
Step over	**F10**
Step out	**SHIFT+F11**
Delete all breakpoints	**CTRL+SHIFT+F9**
Display the Breakpoints window	**CTRL+ALT+B**
Display the Locals window	**CTRL+ALT+V, L**
Display the Immediate window	**CTRL+ALT+I**
Display the Call Stack window	**CTRL+ALT+C**
Display the Threads window	**CTRL+ALT+H**
Start or continue debugging	**ALT+F5**
Show next statement	**ALT+NUM**

As we wrap up this concept, I would like to advise that the person needs to have *SYSADMIN* privileges to start debugging inside SQL Server. This is one of the pre-requisites to play with debugging capability. If the end user aborts the debugging session in the middle, then we will be presented with the below error message:

```
Msg 28102, Level 16, State 1, Line 10
Batch execution is terminated because of debugger request.
```



CONCLUSION

Debugging, though a tough option inside SQL Server, has surely come a long way with the integration with Visual Studio to make SQL Server Management Studio much more powerful. These additions to SSMS have surely increased the productivity of Developers and DBAs alike.

ABOUT THE AUTHOR

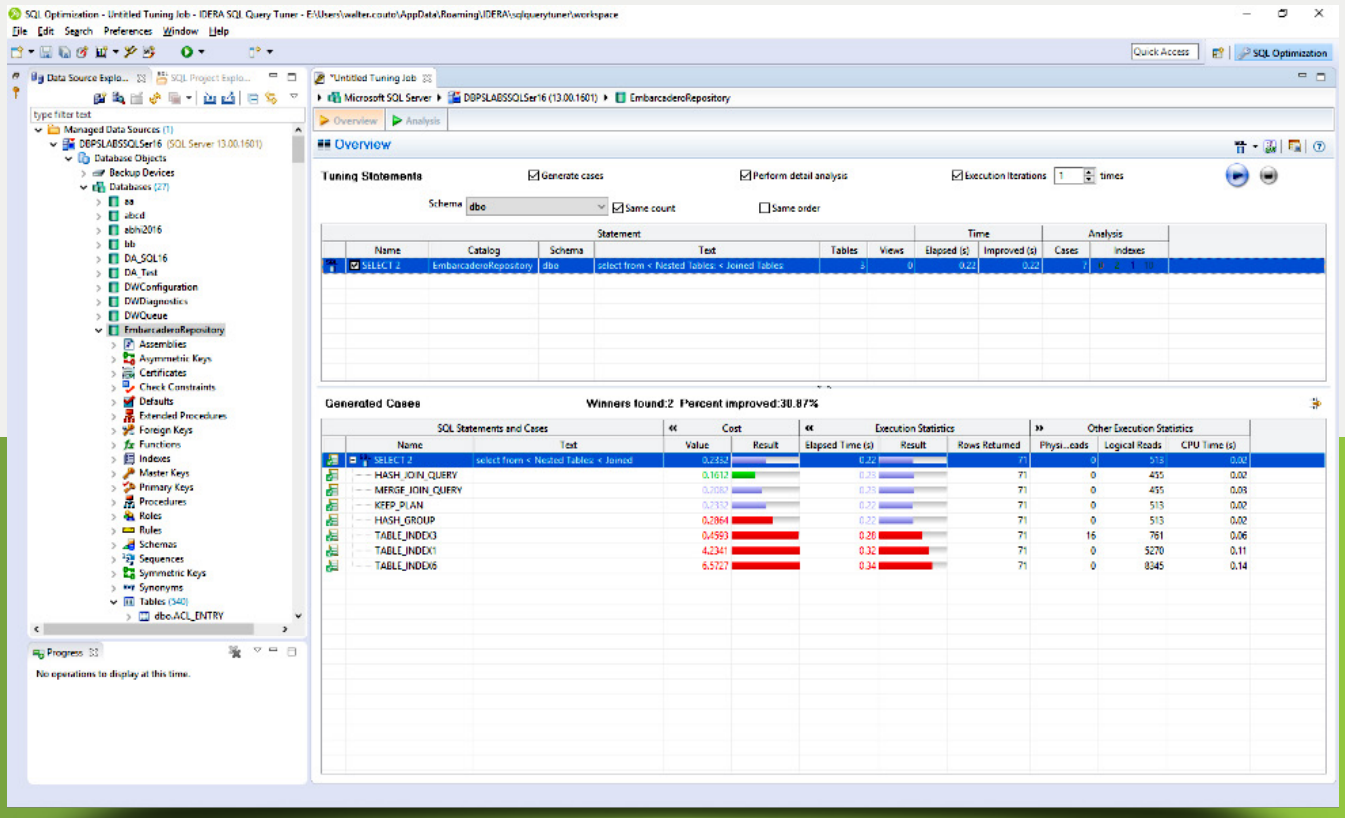
Pinal Dave is a Developer Evangelist. He has authored 11 SQL Server database books, 14 Pluralsight courses and over 2900 articles on the database technology on his blog at http://blog.sqlauthority.com. Along with 10+ years of hands on experience he holds a Masters of Science degree and a number of certifications, including MCTS, MCDBA and MCAD (.NET). His past work experiences include Technology Evangelist at Microsoft and Sr. Consultant at SolidQ.

SQL Query Tuner

TROUBLESHOOT AND OPTIMIZE DATABASE SQL QUERIES

- Visually tune complex SQL queries
- Load test in simulated production environments
- Streamline SQL query tuning for SQL Server

Start for FREE



IDERA

IDERA.com