

# Debug T-SQL execution methods - guidelines to troubleshoot row counts expected

Last updated by | Ricardo Marques | Mar 22, 2023 at 4:36 AM PDT

---

## Contents

- Context
- Understand what the code is doing
- Understand the transaction logic
- Use Extended Events or SQL Profiler
- Elements that might influence the row result - questions to...
- Picking up more rows than the initial ones
- Logging operations
  - Example (stored procedure)
  - Other logging examples - triggers
  - Logging errors during query execution
- Internal reference

## Context

In some cases the customer might point that a certain operation is resulting in less (or more) rows than expected.

Despite of not being entirely our responsibility, we might want to have under our belt some methods to debug T-SQL code when we don't have error codes. In other words, when the problem seems to come all down to logic.

**Note: This all methods to suggest to a customer. The customer is the logic and data owner**

You could be also interested to collect some information about the execution or errors that occur during the execution of each statement.

## Understand what the code is doing

This is the first step of everything. By reading the T-SQL code you might come up with a step by step logic of what the code is doing.

Example:

- Delete from Table A all the records older than 1 year
- insert into table B the result for a SELECT from Table A that INNER JOIN with Table C
- update Table C all the records that satisfies the INNER JOIN condition between Table C and Table A

## Understand the transaction logic

Always have in mind the [atomicity property](#) ☐. This means that, if you try to INSERT 1000 rows, 1000 rows will be inserted. There is no, "I insert 1000 rows and only 10 rows are really written on the table".

The statement "I insert 1000 rows and only 10 rows are really written on the table" can only be true if the INSERT is done on a loop of 10 or less rows. On single batch is impossible

For example, if you do:

```
INSERT INTO TableA
SELECT Column1 from TableB
```

All the rows that come from the SELECT or 0 rows will be inserted. There is no "Only part of the rows selected were inserted"

The same thing happens with multiple statements inside a transaction. Or everything succeeds or everything fails. For example if you have:

```
BEGIN TRANSACTION

INSERT INTO TABLE_A

UPDATE TABLE_C --fails

DELETE FROM TABLE_D

COMMIT TRANSACTION
```

If, for example, the second command fails, the first and second command will Rollback.

## Use Extended Events or SQL Profiler

[Extended events](#) ☐ and [SQL Profiler](#) ☐ can be useful tools when troubleshooting code execution. The main disadvantage is, if there is a lot of activity on the server, you will see several entries that are just making noise. So reading and analyzing the trace can be tricky and exhausting.

Some filters can be applied, but you might end up hiding some details. So choose filters carefully.

Some usual filters that are used: database name, statement, login name.

On SQL Profiler, for capturing executions and respective row counts usually we use:

Template - Tuning

Trace Properties

General | Events Selection

Trace name: Untitled - 1

Trace provider name: azuresqlmi2.public.0d67f5456c3d.database.windows.net,3342

Trace provider type: Microsoft SQL Server 2014 version: 12.0.2000

Use the template: **Tuning**

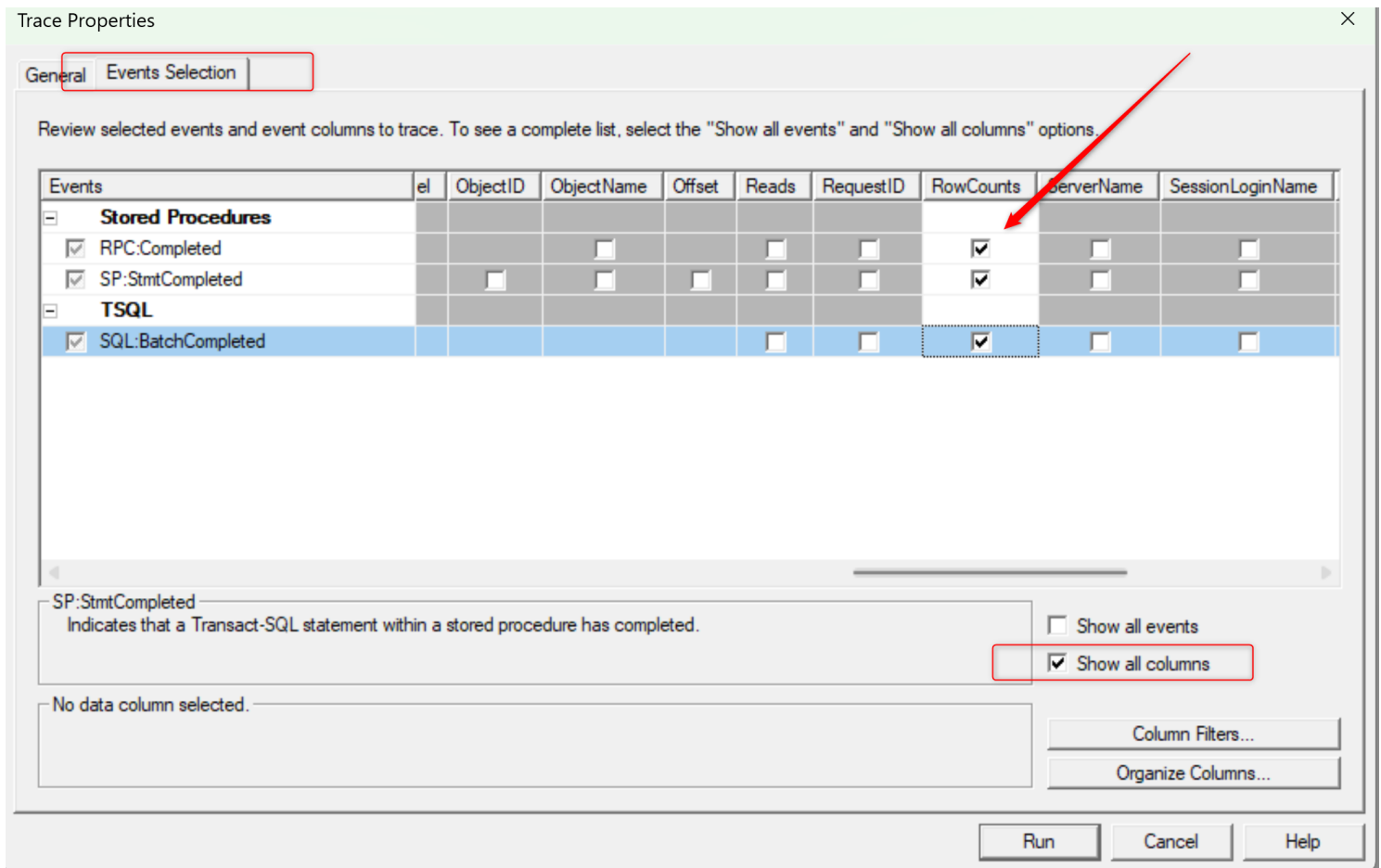
☐ Save to file:   
Set maximum file size (MB): 5   
☒ Enable file rollover   
☐ Server processes trace data

☐ Save to table:   
☐ Set maximum rows (in thousands): 1

☐ Enable trace stop time: 3/14/2023 4:39:45 PM   
☒ Set trace duration (in minutes): 60

Run Cancel Help

This template doesn't have the row count by default. So we have to go to **Events Selection** -> Select **Show all columns** and mark row counts for all Events



You can add other columns that you might find interesting for your specific case.

## Elements that might influence the row result - questions to have in mind

Besides of query logic, other points need to be taken into account when troubleshooting query execution and row counts. In other words, what external elements can influence the final rows.

- Table triggers
- execution of stored procedures in the middle of the code
- application logic (for example is executing on a loop)

## Picking up more rows than the initial ones

When dealing with code logic and data, it might be useful to pick up more that than the initial one.

Some questions that can be done when doing troubleshooting:

- what happens if a WHERE clause is removed
- instead of INNER JOIN, used LEFT (or RIGHT) join. This way you have clear picture of all the records from the LEFT (or RIGHT) relation and all the rows from the LEFT (or RIGHT) relation that don't match with the RIGHT (or LEFT relation)

## Logging operations

When writing code, usually when we are starting to learn how to code (even experienced developers might still do), when debugging code we might come to a point where we started printing messages on a middle of the code. This would help, for example, to know if a variable had a right value, if the execution was reaching part of our code, etc.

In SQL, if the execution is being done through SSMS we can use same method of printing. But if the code is being executed from another client like ADF, we have to come up with a different strategy since Prints are not visible.

### Example (stored procedure)

Lets use an example of a stored procedure. This procedure deletes from a table and inserts on another table.

```
CREATE PROCEDURE usp_test(@customerID int)
as
begin
    delete from tableA

    insert into tableA
    select column1, column2
    from tableB b
    inner join tableC
    b.id = a.id
    where b.customerID = @customerID

end
```

Note that this is only an example. Build your logging according with your case

Before executing the procedure create the table below for logging:

```
create table rowcountlog(
    customerID int,
    oper varchar(30),
    rowcount bigint,
    datehour datetime
)
```

Then create a new procedure that logs each operation, based on the previous one - this way you will have an isolation from other executions

```

alter PROCEDURE usp_test(@customerID int)
as
begin

    declare @rows bigint -- variable for rowcount
    delete from tableA
    --logging input value, operation, rowcount and the date of the operation
    set @rows = @@rowcount
    insert into rowncountlog values (@customerID, 'DELETE', @rows, (select getdate()))

    insert into tableA
    select column1, column2
    from tableB b
    inner join tableC
    b.id = a.id
    where b.customerID = @customerID
    --logging input value, operation, rowcount and the date of the operation
    set @rows = @@rowcount
    insert into rowncountlog values (@customerID, 'INSERT', @rows, (select getdate()))

end


```

Executing the procedure above will write 2 records to the log table - one for the INSERT and one for the DELETE. Make sure that the execution replicates exactly the problem, in other words, execute from the customer application.

This will help answer some questions like:

- how many rows are being affected on each execution?
- what was the row count for the DELETE (or the INSERT)?
- since I'm expecting only two log records for each execution, isn't the procedure running on a loop from the application?

### Other logging examples - triggers

Customer can also create [Triggers](#)  as a method of logging write operations, but they might be trickier to develop and log. Logging only for specific executions might be more helpful.

### Logging errors during query execution

Sometimes getting an error message from an application can be difficult. Sometimes is also not clear if certain commands are actually failing.

One solution is to log the errors into a table. [TRY CATCH](#)  can help you with that

```
-- create a table for saving errors, when they occur

CREATE TABLE Error_logging
    (ErrorDateTime DATETIME,
     ErrorNumber   INT,
     ErrorState    INT,
     ErrorSeverity INT,
     ErrorLine     INT,
     ErrorMessage  VARCHAR(MAX),
    )

GO

-- in the example below we are logging the error on the error logging table if the INSERT fails

BEGIN TRY
    INSERT INTO TableA Values(1,'aaaaa')
END TRY
BEGIN CATCH
    INSERT INTO Error_logging
    VALUES
    (GETDATE(),
     ERROR_NUMBER(),
     ERROR_STATE(),
     ERROR_SEVERITY(),
     ERROR_LINE(),
     ERROR_MESSAGE(),
    );
END CATCH

GO
```

## Internal reference

[2301180010003312](#) 

**How good have you found this content?**



-