T-SQL static code analysis: "NOCOUNT" should be activated on "PROCEDURE" and "TRIGGER" definitions

2 minutes

`NOCOUNT` is by default deactivated (OFF) at server level. It means by default, the server will send to the client the number of rows affected by the SQL query executed which is, in most cases, useless because no one will read this information.

Deactivating this feature will save some network traffic and improve the execution performance of stored procedures and triggers that's why it is recommended to define `SET NOCOUNT ON` at the beginning of the definition of `PROCEDURE`s and `TRIGGER`s, before any query is processed.

This rule raises an issue when `NOCOUNT` is not set or is set to OFF between the beginning of the `PROCEDURE` (or `TRIGGER`) definition and the first statement that is not a `SET`, `IF` or `DECLARE`.

Noncompliant Code Example

```
CREATE PROCEDURE dbo.MyProc
AS
BEGIN
  DECLARE @var INT;
```

```
  SET NOCOUNT OFF; -- Noncompliant; deactivate NOCOUNT
  SELECT COUNT(*) FROM MY_TABLE
END;

CREATE PROCEDURE dbo.MyProc
AS
BEGIN
  -- Noncompliant; SET NOCOUNT is not specified so behaviour
of the procedure execution is based on server configuration
(OFF by default)
  SELECT COUNT(*) FROM MY_TABLE
END;

CREATE PROCEDURE dbo.MyProc
AS
BEGIN
  SELECT COUNT(*) FROM MY_TABLE
  SET NOCOUNT ON -- Noncompliant; SET NOCOUNT is set
after select statement
END;
```

Compliant Solution

```
CREATE PROCEDURE dbo.MyProc(@debug  INT)
AS
BEGIN
  DECLARE @var INT;
  IF @debug = 0
    BEGIN
      SET NOCOUNT ON;
    END
  SELECT COUNT(*) FROM MY_TABLE
END;
```

```
CREATE TRIGGER MyTrigger ON MyTable
AFTER INSERT
AS
BEGIN
  SET NOCOUNT ON;
  [...]
END;
```