T-SQL static code analysis: "CHECK" or "NOCHECK" should be specified explicitly when constraints are activated

2 minutes

When you add a new constraint to a table, (`ALTER TABLE ... ADD CONSTRAINT ...`), `WITH CHECK` is assumed by default, and existing data are automatically validated.

But when you disable/enable an existing constraint, `WITH NOCHECK` is assumed by default, and existing data are no longer trusted. In this case you will face an integrity issue that prevents some rows from being updated, and a performance issue because the query optimizer cannot trust this constraint anymore.

Of course, `WITH CHECK` is obviously preferred, but if `NOCHECK` behavior is desired, it should not be selected by omission, but specified explicitly because `WITH NOCHECK` has such a significant impact. By making `NOCHECK` explicit, the developer documents that this behavior has been selected on purpose.

Note: You can list the existing constraints that are in an untrusted state using:

```
SELECT * FROM sys.foreign_keys WHERE
is_not_trusted = 1;
```

```
SELECT * FROM sys.check_constraints WHERE
is_not_trusted = 1;
```

Noncompliant Code Example

```
-- Create a trusted constraint
ALTER TABLE users ADD CONSTRAINT max_age CHECK
(age < 200) ;

-- Disable the constraint
ALTER TABLE users NOCHECK CONSTRAINT max_age;

-- Enable the constraint
ALTER TABLE users CHECK CONSTRAINT max_age; --
Noncompliant, 'WITH NOCHECK' is the default mode, but is it
really intentional?
```

Compliant Solution

```
-- Create a trusted constraint
ALTER TABLE users ADD CONSTRAINT max_age CHECK
(age < 200) ;

-- Disable the constraint
ALTER TABLE users NOCHECK CONSTRAINT max_age;

-- Enable the constraint
ALTER TABLE users WITH CHECK CHECK CONSTRAINT
max_age;
-- OR
ALTER TABLE users WITH NOCHECK CHECK CONSTRAINT
max_age;
```