http://www.sqlservercentral.com/articles/ANSI_NULLs/69234/
Printed 2010/01/27 05:03AM

Get Your ANSI_NULLs Settings Consistent

By William Talada, 2010/01/07

SET IGNORE_FOR_AS_LONG_AS_POSSIBLE_EVERYTHING_TO_DO_WITH_ANSI_SETTINGS ON

Most likely your databases are working sufficiently well that you don't want to even think about digging into ANSI standards and aligning your settings with the rest of the world. But, if you look in the BOL index for "null values [SQL Server], comparison operators" you'll see an important message stating that ANSI_NULLS will be forced ON in a future release. And believe me, if you haven't looked into this, you probably have lurking bugs in your code.

If you read the remarks underneath the message, you'll see that things get confusing and buggy when comparing null columns and variables. It took me a while to decipher exactly which settings makes everyone happy. I'll show you the five simple steps I've taken to get through this quagmire to get back to producing bug free code on the ANSI bandwagon.

1) First go into SSMS and go into Tools > Options > Query Execution > SQL Server > ANSI and press the button labeled "Reset to Default" so we are working with the same connection settings.

2) Make sure your databases have the correct settings since they affect object creation settings. At the very least, try to get model updated so new databases will be created correctly.

```
-- list databases and their ANSI settings
select
        name,
        case when
         is_ansi_null_default_on = 1
        and is_ansi_nulls_on=1
        and is_ansi_padding_on=1
        and is_ansi_warnings_on=1
        and is_arithabort_on=1
        and is_concat_null_yields_null_on=1
        and is_numeric_roundabort_on=0
        and is_quoted_identifier_on=1 then 'ANSI'
        else 'not'
        end
from
        sys.databases
order by
        name
```

3) Make sure your objects were created with the correct settings by running the next six selects. The tables are the hardest to fix, since they will need scripted and recreated. You can trick the scripter into making a script by making a minor column change such as nullability, then manually changing it back in the generated code. If you don't have the willpower to fix these objects right now, please continue on to steps 4 and 5.

```
select
        name as 'Table Not ANSI',
        uses_ansi_nulls
from
        sys.tables
where
        uses_ansi_nulls <> 1
order by
        name

select
        name as 'Proc Not ANSI',
        objectproperty(object_id, 'ExecIsAnsiNullsOn') as AnsiNullsOn,
        objectproperty(object_id, 'ExecIsQuotedIdentOn') as QuotedIdentOn
from
        sys.procedures
where
        (isnull(objectproperty(object_id, 'ExecIsAnsiNullsOn'),0)=0 or isnull(objectproperty(object_id, 'ExecIsQuotedIdentOn'),0)=0)
and
        name not in (select cast(cast(assembly_method as varbinary(100)) as varchar(100)) from sys.assembly_modules)
order by
        name

select
        name as 'Trigger Not ANSI',
        objectproperty(object_id, 'ExecIsAnsiNullsOn') as AnsiNullsOn,
        objectproperty(object_id, 'ExecIsQuotedIdentOn') as QuotedIdentOn
from
        sys.triggers
where
        (isnull(objectproperty(object_id, 'ExecIsAnsiNullsOn'),0)=0 or isnull(objectproperty(object_id, 'ExecIsQuotedIdentOn'),0)=0)
order by
        name

select
```

```
        name as 'View Not ANSI',
        objectproperty(object_id, 'ExecIsAnsiNullsOn') as AnsiNullsOn,
        objectproperty(object_id, 'ExecIsQuotedIdentOn') as QuotedIdentOn
from
        sys.views
where
        (isnull(objectproperty(object_id, 'ExecIsAnsiNullsOn'),0)=0 or isnull(objectproperty(object_id, 'ExecIsQuotedIdentOn'),0)=0)
order by
        name

select
        name as 'Function Not ANSI',
        objectproperty(object_id, 'ExecIsAnsiNullsOn') as AnsiNullsOn,
        objectproperty(object_id, 'ExecIsQuotedIdentOn') as QuotedIdentOn
from
        sys.objects
where
        (isnull(objectproperty(object_id, 'ExecIsAnsiNullsOn'),0)=0 or isnull(objectproperty(object_id, 'ExecIsQuotedIdentOn'),0)=0)
and
        type = 'FN'
order by
        name

select
        object_name(object_id) as 'Column Not ANSI',
        c.*,
        t.name
        --select distinct t.name
from
        sys.columns c
join
        sys.types t on c.system_type_id = t.system_type_id
where
        is_ansi_padded =0
and
        t.name in ('nchar')
and
        object_name(object_id) <> 'sysfiles1'
```

4) Fix lurking bugs by replacing all "= null" with "is null" since this will work regardless of ANSI settings. Also, replace all "<> null" with "is not null".

```
--Run the following query to return a list of objects that might need changed.
-- Some false positives are returned since initializing with "set @v = null" is a valid ANSI statement.
select distinct 'exec sp_helptext '+object_name(id)
from sys.syscomments
where text like '%= null%'
order by 1
```

5) You can use the following function anytime you need to do a nullable ints comparision in your t-sql. This is only necessary when comparing a variable to a column. This happens quite often with stored procedure parameters in "where" clauses. It would have been nice for this to have been built into t-sql. You may need to make another function called NullableVarcharsMatch.

```
-- compare nullable int variables
-- return true if the value matches or if both parameters are null
create function [dbo].[NullableIntsMatch] (@a int, @b int)
returns Bit
as
-- written by Bill Talada
begin
        -- no need to call this function if @a and @b are both columns
        -- function calls are slow...so if you want to inline the tests, use one of the following lines...
        --same: where ((@a = @b) or (@a is null and @b is null))
        --diff: where ((@a <> @b) or (@a is null and @b is not null) or (@a is not null and @b is null))

        declare @r bit
        if @a = @b or (@a is null and @b is null) set @r=1 else set @r=0
        return (@r)
end
GO
```

Good work! Rest assured your queries will no longer be "missing" rows.