

Strange bug, hard to reproduce – is it known?

Source:

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.programming/2005-01/3485.html>

From: Hugo Kornelis (*hugo_at_pe_NO_rFact.in_SPAM_fo*)

Date: 01/18/05

Date: Tue, 18 Jan 2005 18:45:56 +0100

Hi,

I'm working on a system that uses multiple stored procedures, calling each other and exchanging information through temp tables. The base tables in my system all store two versions of each row; they are accessed through views that expose either the "new" or the "old" row, based on a comparison of settings in a control table with control columns in the tables.

Within this complex code, I have encountered a SQL Server bug. One query suddenly started to return too much rows. The funny thing is: none of the tables used in the query, nor the query itself were changed (I did make changes to other tables, other stored procedure and other queries in this particular stored procedure), yet it did work correctly before.

I tried to make a complete repro script, but failed. I recreated all tables involved in my test database, using the EXACT same columns, constraints and triggers and the EXACT same views. I inserted the data exactly as it is in my development database at the moment that my procedure executes the failing query (in fact, I just inserted calls to `sp_generate_inserts` in the proc, just before the failing query). And yet, the problem does not appear in my test database, but it still persists in my development database.

Since I can't provide a full repro anyway, I then decided to strip the repro script of all irrelevant columns and skip the INSERT statements and post the remaining simplified script here. I guess I just hope that it's a known issue, and that somebody might be able to point me to a MSKB article dealing with this.

Anyway, the table structure is at the end of this post. Here is the query that I'm having problems with:

```
SELECT * --- Using SELECT * for debug purposes only;
        --- real code is INSERT ... SELECT xxx, yyy, ...
FROM #Tmp1 AS #
INNER JOIN dbo.GR AS r
        ON r.GRkey = #.GRkey
```

microsoft.public.sqlserver.programming: Strange bug, hard to reproduce – is it known?

```
AND r.GRCol1 = 'DBX'  
CROSS JOIN dbo.FT AS f  
WHERE #.TmpCol1 = 'FT'  
AND f.FTCol1 IS NOT NULL
```

This query returns 4 rows, whereas it should return 2 rows. If I remove the last line (AND f.FTCol1 IS NOT NULL), it returns 2 rows (as I expect) – but this will obviously not produce the correct results if I start adding rows with NULL to my test data. If I change the last line to read AND f.FTCol1 + " IS NOT NULL the problem is also gone (so that will be my workaround for now). Note that the FTCol1 column is not included in any index!

In case it helps:

```
SELECT *  
FROM #Tmp1 AS #  
INNER JOIN dbo.GR AS r  
    ON r.GRkey = #.GRkey  
    AND r.GRCol1 = 'DBX'  
WHERE #.TmpCol1 = 'FT'
```

returns 1 row;

```
SELECT *  
FROM dbo.FT AS f  
WHERE f.FTCol1 IS NOT NULL
```

returns 2 rows. That's the reason I expected 2 rows from the cross join.

I also tried rewriting as an INNER JOIN, but that didn't help either:

```
SELECT *  
FROM #Tmp1 AS #  
INNER JOIN dbo.GR AS r  
    ON r.GRkey = #.GRkey  
    AND r.GRCol1 = 'DBX'  
INNER JOIN dbo.FT AS f  
    ON f.FTCol1 IS NOT NULL  
WHERE #.TmpCol1 = 'FT'
```

still 4 rows....

Here's the DDL to create the tables and views used in the failing query:

```
CREATE TABLE dbo.Conv  
    (ConvNr int NOT NULL  
    ,Actief varbinary(85) NULL  
    ,CONSTRAINT pk_Conv  
    PRIMARY KEY NONCLUSTERED(ConvNr))  
GO  
CREATE CLUSTERED INDEX ix_Conv ON dbo.Conv(Actief)  
GO  
CREATE TABLE dbo.f_FT (Vrs char(1) NOT NULL,  
    FTkey varchar(40) NOT NULL,  
    FTCol1 varchar(45) NULL,  
    Daw char(1) NOT NULL,  
    ConvNr int NULL,
```

Strange bug, hard to reproduce – is it known?

microsoft.public.sqlserver.programming: Strange bug, hard to reproduce – is it known?

```
        CONSTRAINT pk_FT
        PRIMARY KEY CLUSTERED(FTkey, Vrs))

GO
CREATE VIEW dbo.FT
AS
SELECT f.FTkey,
       f.FTCol1
FROM dbo.f_FT AS f
LEFT JOIN dbo.Conv AS c
      ON c.Actief = SUSER_SID()
WHERE f.Daw = 'j'
AND f.Vrs = CASE WHEN f.ConvNr = c.ConvNr THEN 'n' ELSE 'o'
END
GO
CREATE TABLE dbo.f_GR (Vrs char(1) NOT NULL,
                       GRkey varchar(400) NOT NULL,
                       GRCol1 varchar(20) NULL,
                       Daw char(1) NOT NULL,
                       ConvNr int NULL,
                       CONSTRAINT pk_GR
                       PRIMARY KEY CLUSTERED(GRkey, Vrs))

GO
CREATE VIEW dbo.GR
AS
SELECT f.GRkey,
       f.GRCol1
FROM dbo.f_GR AS f
LEFT JOIN dbo.Conv AS c
      ON c.Actief = SUSER_SID()
WHERE f.Daw = 'j'
AND f.Vrs = CASE WHEN f.ConvNr = c.ConvNr THEN 'n' ELSE 'o'
END
GO
```

Best, Hugo

```
--
(Remove _NO_ and _SPAM_ to get my e-mail address)
```