

## Re: insert rows : generating sequence numbers

**Source:**

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.programming/2004-04/0961.html>

---

**From:** Tore Bostrup (*newspost\_at\_bostrup.us*)

**Date:** 04/04/04

Date: Sun, 4 Apr 2004 11:56:22 -0400

A few other options for detecting "holes" in a sequence range (the first two are courtesy David L. Penton):

\*\* Alternative 1:

There is a trick to this one. If you had that list in a temp table structured like so, this gets the first # that is not present:

(if you only have 0, NULL then return 1 and you are done)

```
CREATE TABLE #TEMP (  
    val int NOT NULL  
)
```

Then this tells you the answer:

```
SELECT TOP 1  
    COUNT(DISTINCT b.val)  
FROM  
    #TEMP a  
inner join  
    #TEMP b  
on  
    a.val >= b.val  
    -- this *should* really be something that  
    -- restricted a.val and b.val to positive integers  
    AND 0 NOT IN (a.val, b.val)  
group by  
    a.val  
having  
    sum(DISTINCT b.val) <> a.val * (a.val + 1) / 2
```

which is based on the fact that the sum of the nth natural number in a steadily increasing sequence (1, 2, 3, ..., n)

$SUM(1, 2, 3, \dots, n) = n * (n + 1) / 2!$

microsoft.public.sqlserver.programming: Re: insert rows : generating sequence numbers

(based on Pascal's Triangle, or should I say one of the diagonals of it)

This gets the first one that is not present.

\*\* Alternative 2:

There are of course ways to use a TEMP table like that in a loop to find the values that are not present. It wouldn't be too hard to work that into a UDF as well.

Something like:

```
/* OTTOMH and not optimized */
CREATE FUNCTION udfFindIntegerHoles
RETURN TABLE (
    val int NOT NULL
)
BEGIN

DECLARE @vals TABLE (
    id int identity(1, 1) NOT NULL
    val int null
)

DECLARE @output TABLE (
    val int not null
)

DECLARE @id int, @max int

SET @id = 0

SELECT @max = MAX(a.id)
FROM myTable

WHILE @id <= @max
BEGIN
    INSERT INTO @vals (val) VALUES (NULL)
    SET @id = @@IDENTITY
    UPDATE a
    SET a.val = b.id
    FROM
        @vals a
        , (
            SELECT @id
            WHERE NOT EXISTS (
                SELECT NULL FROM myTable z
                WHERE z.id = @id
            )
        ) b
END
```

Re: insert rows : generating sequence numbers

```
INSERT INTO @output
SELECT val FROM @vals WHERE val IS NOT NULL
```

```
RETURN @output
```

```
END
```

```
** Alternative 3:
```

```
-- Set up the temporary test table:
```

```
CREATE TABLE #TEMP (
    val int NOT NULL
```

```
)
```

```
GO
```

```
--INSERT INTO #TEMP VALUES (1)
```

```
INSERT INTO #TEMP VALUES (2)
```

```
INSERT INTO #TEMP VALUES (3)
```

```
INSERT INTO #TEMP VALUES (5)
```

```
INSERT INTO #TEMP VALUES (6)
```

```
INSERT INTO #TEMP VALUES (7)
```

```
INSERT INTO #TEMP VALUES (12)
```

```
INSERT INTO #TEMP VALUES (13)
```

```
INSERT INTO #TEMP VALUES (14)
```

```
INSERT INTO #TEMP VALUES (19)
```

```
INSERT INTO #TEMP VALUES (21)
```

```
--Script :
```

```
DECLARE @Holes Table (HoleFrom int Not Null, HoleTo int Null)
```

```
DECLARE @MissingVal TABLE (val Int Not NULL)
```

```
DECLARE @HoleFrom Int, @HoleTo Int, @iMissing Int
```

```
DECLARE curMissingRanges CURSOR
```

```
LOCAL FAST_FORWARD
```

```
FOR Select HoleFrom, HoleTo From @Holes
```

```
-- Find the beginning of each hole:
```

```
Insert Into @Holes (HoleFrom)
```

```
-- The first part covers a hole at the beginning of the range (in this case
```

```
1)
```

```
SELECT 1
```

```
WHERE Not Exists (Select 1 from #TEMP WHERE val = 1)
```

```
UNION
```

```
-- This parts returns the "first non-existent value"
```

```
-- for each break in the sequence, including the last value
```

```
SELECT A.val + 1 as HoleFrom
```

```
FROM #TEMP as A -- "All" Instances
```

```
WHERE Not Exists (SELECT 1 FROM #TEMP as F WHERE F.val = A.val + 1)
```

microsoft.public.sqlserver.programming: Re: insert rows : generating sequence numbers

```
-- Obtain the "last non-existent value" for each range
-- leaves NULL for range that starts at Max(val) + 1:
UPDATE @Holes
SET HoleTo = A.val - 1
FROM #TEMP as A -- "All" Instances
  INNER JOIN @Holes as H
  ON A.val - 1 >= H.HoleFrom
WHERE Not Exists (SELECT 1 FROM #TEMP as F WHERE F.val = A.val - 1)

-- Parse the "hole" ranges (I know - it's a cursor... :-<):
OPEN curMissingRanges

FETCH NEXT FROM curMissingRanges
INTO @HoleFrom, @HoleTo

WHILE @@FETCH_STATUS = 0 BEGIN
  SET @iMissing = @HoleFrom
  -- First, handle the Max(val) + 1 situation
  IF @HoleTo IS NULL INSERT INTO @MissingVal (val) VALUES (@iMissing)
  ELSE BEGIN
    -- Calculate and store each value in "hole" range
    While @iMissing <= @HoleTo BEGIN
      INSERT INTO @MissingVal (val) VALUES (@iMissing)
      SET @iMissing = @iMissing + 1
    END
  END
  FETCH NEXT FROM curMissingRanges
  INTO @HoleFrom, @HoleTo
END

-- Return a record set with the "missing" values, including Max(val) + 1
Select * From @MissingVal
```

```
"Tenaya" <ct@ct.ct> wrote in message
news:u0Nt5ujGEHA.576@TK2MSFTNGP11.phx.gbl...
> John,
>
> Well, never say die :-)
>
> If you are willing to create an auxiliary table, then there is a
> set-oriented approach.
>
> For the sake of simplicity and clarity, I've eliminated the character
> manipulation part of the code, and have simply concentrated on what I
> believe is the difficult part ... namely, backfilling sequence numbers.
I've
> also created views to (substantially) simplify the code. Views are never
> necessary, but can often times be very helpful in understanding the logic
> ... and, I think, particularly in this case:
>
> create table Numbers (n int NOT NULL primary key)
```

Re: insert rows : generating sequence numbers

```
> go
>
> insert into Numbers values (0)
> insert into Numbers values (1)
> insert into Numbers values (2)
> insert into Numbers values (3)
> insert into Numbers values (4)
> insert into Numbers values (5)
> insert into Numbers values (6)
> insert into Numbers values (7)
> insert into Numbers values (8)
> insert into Numbers values (9)
> go
>
> create table Table1 (Col1 int NOT NULL unique)
> create table Table2
> (
> Col1 int NOT NULL,
> Col2 int NOT NULL
> )
> go
>
> insert into Table2 values (2, 2)
> insert into Table2 values (4, 4)
>
> insert into Table1 values (3)
> insert into Table1 values (5)
> insert into Table1 values (100)
> insert into Table1 values (111)
> go
>
> create view linear_Table1 as
> select Col1, s = (select count(*)
> from Table1 as t2
> where t2.Col1 < t1.Col1)
> from Table1 as t1
> go
>
> create view backfill_Numbers as
> select backfill = n
> from Numbers
> where not exists (select *
> from Table2 as t2
> where n = t2.Col2)
> go
>
> create view seq_backfill_Numbers as
> select backfill,
> n = (select count(*)
> from backfill_Numbers as b2
> where b2.backfill < b1.backfill)
```

```
> from backfill_Numbers as b1
> go
>
> insert into Table2
> select t1.Col1, b.backfill
> from linear_Table1 as t1
> join seq_backfill_Numbers as b
> on (t1.s = b.n)
> go
>
> select * from Table2
> go
>
> Chief Tenaya
>
>
> "John A Grandy" <johnagrandy-at-yahoo-dot-com> wrote in message
> news:%23MsAnmhGEHA.712@tk2msftngp13.phx.gbl...
>> Regarding generating sequence numbers ... if the table inserted into
>> (Table2) has pre-existing rows , then the problem becomes more complex
...
>>
>> I need to insert into Table2 from Table1.
>>
>> Table1 contains some rows where Table1.Col1 char(7) matches the pattern
>> '555mmmm' -- but 'mmmm' is not in an increment-by-1 sequence (sample
> data
>> : '5550001' , '5550017' , '5550100' )
>>
>> Table2 contains some rows where Table2.Col2 char(7) matches the pattern
>> '666nnnn' -- but 'nnnn' is not in an increment-by-1 sequence (sample
> data
>> : '5550001' , '5550017' , '5550100' )
>>
>> I need to insert one Col2 = '666nnnn' row into Table2 for each Col1 =
>> '555mmmm' row in Table1 -- and I need to sequentially fill the Col2
>> sequence gaps , in the order given by Col1
>>
>> sample data :
>>
>> Table2.Col2
>> '6660002'
>> '6660004'
>>
>> Table1.Col1
>> '5550003' -- want to translate to Table2.Col2 = '6660000'
>> '5550005' -- want to translate to Table2.Col2 = '6660001'
>> '5550100' -- want to translate to Table2.Col2 = '6660003'
>> '5550111' -- want to translate to Table2.Col2 = '6660005'
>>
>>
```

microsoft.public.sqlserver.programming: Re: insert rows : generating sequence numbers

> >  
>  
>