

What's New in TSQL 2012

PRESENTED BY
STEVE STEDMAN



<http://SteveStedman.com>

Follow me on Twitter @SqlEmt

Presenter: Steve Stedman

- DBA/Consultant/Trainer/Speaker/Writer
 - Been using SQL Server since 1991
 - Taught SQL Server classes at WWU
 - SQL Server consultant
 - Writer contributing to a new SQL Server book – Tribal SQL
- Developer of the Database Health Project
 - <http://DatabaseHealth.SteveStedman.com>
- Working at Emergency Reporting as CTO
- Volunteer Firefighter and EMT
- <http://SteveStedman.com> for more information.

<http://SteveStedman.com>

Follow me on Twitter @SqlEmt

SQL Server 2012

- Introduced in Spring of 2012
 - SP1 Released November 6, 2012 with some additional features
- Many new features and enhancements
 - Always On Column Store Index
 - And many others
 - Not part of this presentation
- This presentation will focus on the TSQL specific new features enhancements.
- TSQL Analytic Functions are not part of this presentation. See my blog for more info on 2012 Analytic Functions.

New Features Overview

- OVER Clause Enhancements
 - ROWS PRECEDING, FOLLOWING, UNBOUNDED
 - RANGE PRECEDING, FOLLOWING, UNBOUNDED
- IIF – Immediate IF or Inline IF (from access)
- CHOOSE (from access)
- OFFSET / FETCH
- FORMAT
- CONCAT
- SEQUENCE

More New Features Overview

- sp_describe_first_result_set
- New Date and Time Functions
- Conversion Functions
 - PARSE, TRY_PARSE, TRY_CONVERT
- THROW exception

The Pre-2012 OVER Clause

- The OVER clause before SQL Server 2012...
- Really handy to perform aggregates over a different range than your standard grouping.

```
select *,  
    avg(Revenue) OVER (PARTITION by DepartmentID) as AverageRevenue,  
    sum(Revenue) OVER (PARTITION by DepartmentID) as TotalRevenue  
from REVENUE  
order by departmentID, year;
```

OVER Clause Enhancements

- **ROWS PRECEDING, FOLLOWING, UNBOUNDED**
 - Rows refers to the current row and those before or after based on preceding or following.
- **RANGE PRECEDING, FOLLOWING, UNBOUNDED**
 - Range means all values in the current range and those before or after.

Year	DepartmentID	Revenue	RowsCumulative	RangeCumulative
1	2003	1	10300	10300
2	2004	1	10000	20300
3	2005	1	20000	40300
4	2005	1	10000	50300
5	2006	1	40000	90300
6	2007	1	70000	160300
7	2008	1	50000	210300

ROWS / RANGE PRECEDING, FOLLOWING, UNBOUNDED

- ROWS or RANGE- specifying rows or range
- PRECEDING – get rows before the current one
- FOLLOWING – get rows after the current one
- UNBOUNDED – get all before or after
- CURRENT ROW

```
sum(Revenue) OVER (PARTITION by DepartmentID
                   ORDER BY [YEAR]
                   ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) as Prev3
```

```
sum(Revenue) OVER (PARTITION by DepartmentID
                   ORDER BY [YEAR]
                   ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) as BeforeAndAfter
```

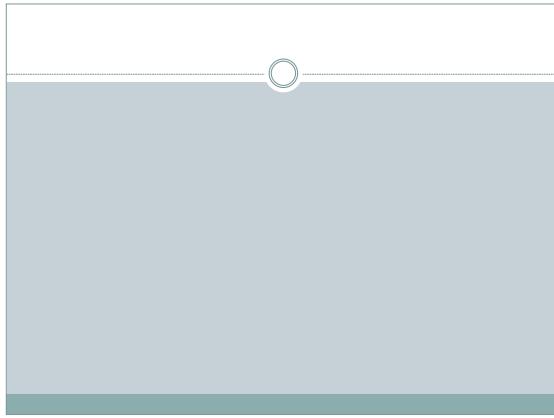
ROWS/RANGE PRECEDING, FOLLOWING, UNBOUNDED - Demo

```
--ROWS PRECEDING
-- http://stevededman.com/?p=1454
-- sum(Revenue) over a range of rows before a trailing 3 year period
select Year, DepartmentID, Revenue
    ,sum(Revenue) OVER (PARTITION by DepartmentID
                        ORDER BY [YEAR]
                        ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) as Prev3
from REVENUE order by departmentID, year;

--ROWS PRECEDING
select Year, DepartmentID, Revenue,
    ,sum(Revenue) OVER (PARTITION by DepartmentID
                        ORDER BY [YEAR]
                        ROWS BETWEEN 5 PRECEDING AND 2 PRECEDING) as Prev5to2
from REVENUE order by departmentID, year;

-- ROWS FOLLOWING
-- http://stevededman.com/?p=1454
select Year, DepartmentID, Revenue,
    ,sum(Revenue) OVER (PARTITION by DepartmentID
                        ORDER BY [YEAR]
                        ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING) as Next3
from REVENUE order by departmentID, year;

--ROWS PRECEDING
select Year, DepartmentID, Revenue,
    ,sum(Revenue) OVER (PARTITION by DepartmentID
                        ORDER BY [YEAR]
                        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) as BeforeAndAfter
from REVENUE order by departmentID, year;
```



IIF

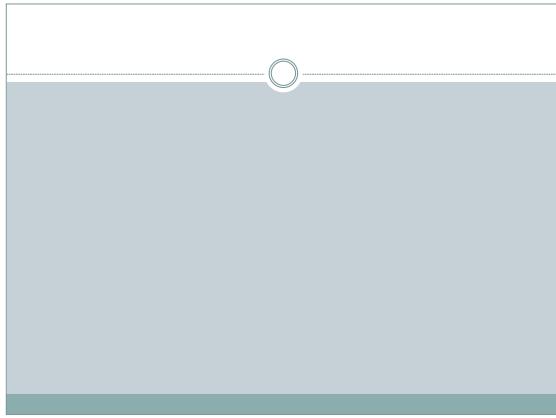
- Inline IF
- Similar to MS Access
- Similar to the case statement but easier to write and it has only one condition.
- Evaluates boolean expression and returns one of two results based on that boolean

IIF(boolean_expression, true_value, false_value)

- Performance very similar between IIF and CASE
- IIF simplifies the code over using a CASE statement

IIF Example

```
-- now the same functionality using IIF and simplifying the code
-- http://stevestedman.com/?p=1578
select Year, DepartmentID, Revenue, AverageRevenue,
    iif(Revenue > AverageRevenue, 'Better Than Average', 'Not') as Ranking
from (select Year, DepartmentID, Revenue
```



CHOOSE

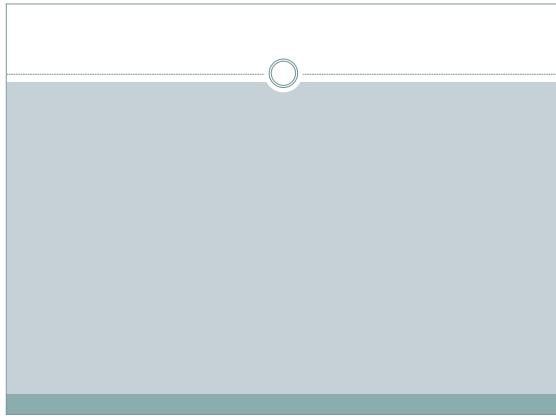
- Function that returns the item at a specific index.
- CHOOSE(index, val_1, val_2, val_3, ...)
- If the index is greater than the number of values or less than 1 it returns NULL
- Easier than CASE in some examples

CHOOSE – Example

```

declare @corners as int = 6
SELECT choose(@corners, 'point', 'line', 'triangle', 'square',
              'pentagon', 'hexagon', 'heptagon', 'octagon')

-- the old way using case.
SELECT CASE @corners
      WHEN 1 THEN 'point'
      WHEN 2 THEN 'line'
      WHEN 3 THEN 'triangle'
      WHEN 4 THEN 'square'
      WHEN 5 THEN 'pentagon'
      WHEN 6 THEN 'hexagon'
      WHEN 7 THEN 'heptagon'
      WHEN 8 THEN 'octagon'
      else NULL
END;
  
```



OFFSET / FETCH

- Its about time....
- Similar to `LIMIT` in MySQL and `ROWNUM` in Oracle, but better
- Used for data paging
 - Easier than the CTE method of doing the same (pre SQL 2012)
 - Faster than the CTE method

**Stop trying to make
FETCH happen**

It's NOT going to happen

OFFSET / FETCH – Example

```
-- the new way to do paging with OFFSET and FETCH
SELECT *
  FROM person.Person
 ORDER BY LastName, FirstName, MiddleName
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY;
```

OFFSET / FETCH Notes

- You MUST have an order by
- You can use hard coded or dynamic values for the rows
- ROW[S] the S is optional on ROWS
- Common Questions
 - What if a row is inserted or deleted between paging queries using OFFSET and FETCH?

FORMAT

- Format strings
- Format dates
- Format currency
- and more



FORMAT

- FORMAT(value, format, [culture])

- Value is the thing to be formatted
- Format specifies how we want it to look
- Optional Culture specifies the specific language/locale used for formatting.

- Easier than CONVERT

```
DECLARE @Revenue MONEY = 314159.26
SELECT '$' + CONVERT(VARCHAR(32),@Revenue,1); Old Way
SELECT FORMAT(@Revenue,'C'); New Way
```

Formatting Types

- A valid .NET Framework format string
- C = Currency
- D = Date
- X = hexadecimal

```
-- old way
DECLARE @Revenue MONEY = 314159.26
SELECT '$' + CONVERT(VARCHAR(32),@Revenue,1);

-- now with format
DECLARE @Revenue MONEY = 314159.26
SELECT FORMAT(@Revenue,'C');
SELECT FORMAT(getdate(), 'd');
SELECT FORMAT(1234, 'X');
```

FORMAT – Culture

- If the *culture* argument is not provided, then the language of the current session is used

- Server default
- SET LANGUAGE

- Examples

- En-us, fr-fr, de-de, jp-jp

```
-- using the culture parameter
DECLARE @Revenue MONEY = 314159.26
SELECT FORMAT(@Revenue,'c','en-us') as English;
SELECT FORMAT(@Revenue,'c','fr-fr') as French;
SELECT FORMAT(@Revenue,'c','de-de') as German;
SELECT FORMAT(@Revenue,'c','ja-JP') as Japanese;
```

FORMAT - Demo

```
-- old way
DECLARE @Revenue MONEY = 314159.26
SELECT 'S' + CONVERT(VARCHAR(32),@Revenue,1);
-- now with format
DECLARE @Revenue MONEY = 314159.26
SELECT FORMAT(@Revenue,C');

-- other examples
SELECT FORMAT(getdate(), 'd');
SELECT FORMAT(1234, 'X');

-- custom format values
SELECT FORMAT(getdate(), 'MMMM dd, yyyy (dddd)');

-- using the culture parameter
DECLARE @Revenue MONEY = 314159.26
SELECT FORMAT(@Revenue,'c','en-us') as English;
SELECT FORMAT(@Revenue,'c','fr-fr') as French;
SELECT FORMAT(@Revenue,'c','de-de') as German;
SELECT FORMAT(@Revenue,'c','ja-JP') as Japanese;
```

CONCAT

- Concatenates data
- Easier than using + because all types are cast to strings
- CONCAT (string1, string2 [, stringN])
- Output is a string, input is more than one string.
- Forces conversion to string
 - PRINT 'Current Time ' + GETDATE()
 - PRINT CONCAT('Current Time ', GETDATE())

CONCAT – Demo

```
-- the old way
SELECT 1 + 'two' + 3.0 + 'four' -- fails without a cast

SELECT cast(1 as varchar(1024)) + 'two' + cast(3.0 as varchar(1024)) + 'four'

-- CONCAT in TSQL 2012
SELECT CONCAT(1, 'two', 3.0, 'four');

-- another example
SELECT 'uniqueidentifier' = ' + NEWID(); -- fails
SELECT CONCAT('uniqueidentifier' = ' + NEWID());

-- PRINT CONCAT TRICK
PRINT 'Time' + GETDATE(); -- fails
PRINT 'Time' + CAST(GETDATE() AS VARCHAR(30));
PRINT CONCAT('Time ', GETDATE());
```

Sequence – Take a Number

- Take a number
- Take one or take several
- For those familiar with Oracle/PLSQL, sequence has been available for years



SEQUENCE

- User-defined object that generates a sequence of numeric values
- Specify the following
 - Start
 - Increment
 - Min Value, Max Value
 - Cycle / No Cycle – starts of when the Max value is hit or not
 - Cache
- Alternative to Identity

SEQUENCE – Details

- Like identity, but you can grab it before you do an insert
- Can be used shared in more than one table
- Defaults to a BIGINT if size not specified
- Gaps
 - Other people / processes can use your SEQUENCE with you knowing it, this will cause gaps
 - Rolled back transactions that ask for the NEXT VALUE and don't use it may cause gaps.

Use a Sequence Instead of Identity When...

- The application requires a number before the insert is run.
- Sharing a single series of numbers between multiple tables or multiple columns within a table is needed.
- The application must restart the number series at some point.
- The application requires sequence values to be sorted by another field. The NEXT VALUE FOR function can apply the OVER clause to the function call.
- You need to change the specification of the sequence, such as the increment value.

SEQUENCE - Syntax

```
CREATE SEQUENCE [schema_name . ] sequence_name
[ AS [ built_in_integer_type | user-defined_integer_type ] ]
[ START WITH <constant> ]
[ INCREMENT BY <constant> ]
[ { MINVALUE [ <constant> ] } | { NO MINVALUE } ]
[ { MAXVALUE [ <constant> ] } | { NO MAXVALUE } ]
[ CYCLE | { NO CYCLE } ]
[ { CACHE [ <constant> ] } | { NO CACHE } ]
[ ; ]
```

SEQUENCE – Demo

```
CREATE SEQUENCE ordersKeySequence
AS int
START WITH 100
INCREMENT BY 1 ;

CREATE TABLE Orders
(OrderID int PRIMARY KEY,
Name varchar(20) NOT NULL,
Qty int NOT NULL);
GO
-- Insert three records
INSERT Orders (OrderID, Name, Qty)
VALUES (NEXT VALUE FOR ordersKeySequence, 'Hat', 2);
INSERT Orders (OrderID, Name, Qty)
VALUES (NEXT VALUE FOR ordersKeySequence, 'Shirt', 1);
INSERT Orders (OrderID, Name, Qty)
VALUES (NEXT VALUE FOR ordersKeySequence, 'Shoes', 1);
GO
-- View the table
SELECT * FROM Orders ;
```



sp_describe_first_result_set

- Returns metadata for the result set returned from a query.
- An alternative to sp_columns

sp_describe_first_result_set – Demo

```
exec sp_describe_first_result_set  
N'SELECT * FROM MyTable';
```

is_hidden	column_ordinal	name	is_nullable	system_type_id	system_type_name	max_length	precision	scale	collation_name
1	0	1	DepartmentID	56	int	4	10	0	NULL
2	0	2	Revenue	56	int	4	10	0	NULL
3	0	3	Year	56	int	4	10	0	NULL
4	0	4	Name	167	varchar(1024)	1024	0	0	SQL_Latin1_General_CI_AS



New Date and Time Functions

- DATEFROMPARTS
- TIMEFROMPARTS
- DATETIMEFROMPARTS
- DATETIME2FROMPARTS
- SMALLDATETIMEFROMPARTS
- DATETIMEOFFSETFROMPARTS
- EOMONTH

DATEFROMPARTS

DATEFROMPARTS (year, month, day)

• Arguments

- Year Integer expression specifying a year.
- Month Integer expression specifying a month, from 1 to 12.
- Day Integer expression specifying a day.

- Always Year – Month – Day order independent of language or location

TIMEFROMPARTS

TIMEFROMPARTS (hour, minute, seconds, fractions, precision)

• Arguments

- Hour Integer expression specifying hours.
- Minute Integer expression specifying minutes.
- Seconds Integer expression specifying seconds.
- Fractions Integer expression specifying fractions.
- Precision Integer literal specifying the precision of the **time** value to be returned.

DATETIMEFROMPARTS

DATETIMEFROMPARTS (year, month, day, hour, minute, seconds, milliseconds)

- **Arguments**

- Year Integer expression specifying a year.
- Month Integer expression specifying a month.
- Day Integer expression specifying a day.
- Hour Integer expression specifying hours.
- Minute Integer expression specifying minutes.
- Seconds Integer expression specifying seconds.
- Milliseconds Integer expression specifying milliseconds.

DATETIME2FROMPARTS

DATETIME2FROMPARTS (year, month, day, hour, minute, seconds, fractions, precision)

- **Arguments**

- Year Integer expression specifying a year.
- Month Integer expression specifying a month.
- Day Integer expression specifying a day.
- Hour Integer expression specifying hours.
- Minute Integer expression specifying minutes.
- Seconds Integer expression specifying seconds.
- Fractions Integer expression specifying fractions.
- Precision Integer literal specifying the precision of the datetime2 value to be returned.

SMALLDATETIMEFROMPARTS

● **SMALLDATETIMEFROMPARTS(year,month,day,hour,minute)**

● Same idea as the others....

DATETIMEOFFSETFROMPARTS

- **DATETIMEOFFSETFROMPARTS** (year, month, day, hour, minute, seconds, fractions, hour_offset, minute_offset, precision)
- *hour_offset* - Integer expression specifying the hour portion of the time zone offset.
- *minute_offset* - Integer expression specifying the minute portion of the time zone offset.
- *Precision* - Integer literal specifying the precision of the value to be returned.
 - Precision can be a value of 0 to 7 which specifies the precision of the fractional part of the seconds.

EOMONTH

EOMONTH (start_date [, month_to_add])

- Returns the last day of the month that contains the specified date, with an optional offset.
- **Arguments**
 - *start_date* Date expression specifying the date for which to return the last day of the month.
 - *month_to_add* Optional integer expression specifying the number of months to add to *start_date*.

Date and Time– Demo

```
-- DATETIMEPARTS(year,day)
-- No del keyword
SELECT DatePart(Year,GetDate()),Convert(varchar(1),Year)-'.'+Convert(varchar(1),Day)+'.'
-- new TSQL DATETIMEPARTS
DECLARE @dt datetime = '2012-01-01 12:00:00.123456789'
--TIMEFROMPARTS(hour,minute,second,fraction,precision)
SELECT TimeFromParts(0,12,0,45,45) AS t1
--DATETIMEFROMPARTS (year,month,day,hour,minute,second,millisecond)
DECLARE @dt datetime = '2012-01-01 12:00:00.123456789'
--DATETIMEFROMPARTS (year,month,day,hour,minute,second,fraction,precision)
SELECT DateTimeFromParts(2012, 1, 1, 12, 0, 0, 45, 45) AS t2
-- Error
-- The precision is smaller than the fraction specified
SELECT DateTimeFromParts(2012, 1, 1, 12, 0, 0, 45, 45) AS t3
--SMALLDATETIMEFROMPARTS (year,month,day,hour,minute,second)
SELECT SmallDateTimeFromParts(0, 0, 0, 0, 0)
--DATETIMEOFFSETFROMPARTS ( year, month, day, hour, minute, seconds, fractions, hour_offset, minute_offset, precision )
SELECT DateFromOffsetFromParts(2012, 1, 1, 12, 0, 0, 45, 45, 12, 30) AS t4
--Provides option for DATETIMEOFFSETFROMPARTS
SELECT DateFromOffsetFromParts(2012, 1, 1, 12, 0, 0, 45, 45, 12, 30) AS t5
SELECT DateFromOffsetFromParts(2012, 1, 1, 12, 0, 0, 45, 45, 12, 30) AS t6
SELECT DateFromOffsetFromParts(2012, 1, 1, 12, 0, 0, 45, 45, 12, 30) AS t7
SELECT DateFromOffsetFromParts(2012, 1, 1, 12, 0, 0, 45, 45, 12, 30) AS t8
SELECT DateFromOffsetFromParts(2012, 1, 1, 12, 0, 0, 45, 45, 12, 30) AS t9
SELECT DateFromOffsetFromParts(2012, 1, 1, 12, 0, 0, 45, 45, 12, 30) AS t10
--EOMONTH
DECLARE @dt DATETIME
SET @dt = '2012-01-01 12:00:00.123456789'
SELECT EOMONTH( @dt,0 ) AS Result;
SELECT EOMONTH( @dt,1 ) AS Result;
```

I don't expect you to read this slide, just copy and paste into SSMS.

Conversion Functions

- **PARSE**

`PARSE (string_value AS data_type [USING culture])`

- **TRY_PARSE**

`TRY_PARSE (string_value AS data_type [USING culture])`

- **TRY_CONVERT**

`TRY_CONVERT (data_type [(length)], expression [, style])`

style accepts the same values as the style parameter of the `CONVERT` function

Conversion Functions Demo

```
-- PARSE ( string_value AS data_type [ USING culture ])
SELECT parse('2012-06-06' as date);

-- this is an invalid date
SELECT parse('2012-06-9' as date);

-- with culture
SELECT parse('06-06-2012' as date USING 'en-us');
SELECT parse('06-06-2012' as date USING 'fr-fr');

-- TRY_PARSE ( string_value AS data_type [ USING culture ]) similar to parse, but null is returned rather than throwing an error for invalid dates
SELECT try_parse('2012-06-06' as date);

-- this is an invalid date
SELECT try_parse('2012-06-9' as date);

-- now with culture
SELECT try_parse('06-06-06' as date USING 'en-us');
SELECT try_parse('06-06-06' as date USING 'fr-fr');
SELECT try_parse('2012-06-06' as date USING 'fr-fr');
-- sometimes invalid 6th day of 6th month in French
SELECT try_parse('06-06-06' as date USING 'fr-fr');

-- TRY_CONVERT ( data_type [ ( length ), expression [, style ] )
SET DATEFORMAT dmy;
SELECT TRY_CONVERT(datetime, '12/31/2010') AS Result;

SELECT CASE WHEN TRY_CONVERT(float, 'test') IS NULL
    THEN Cast failed! ELSE Cast succeeded! END AS Result;

SELECT *,try_convert(float,Name) as newValue
FROM MyTable;
```

THROW exception

- Replaces RAISERROR
- Allows for error handling, then THROW of the original error
- Percolate errors up

THROW Demo

```

SELECT 1/0

CREATE TABLE ErrorLog([ErrTime] datetime, [Severity] varchar(max), [ErrMsg] varchar(max))

-- Prior to SQL 2012
BEGIN TRY
    DECLARE @Number int = 1 / 0;
END TRY
BEGIN CATCH
    -- Log the error info, then re-throw it
    INSERT INTO ErrorLog VALUES(GETDATE(), ERROR_SEVERITY(), ERROR_MESSAGE());
    DECLARE @msg NVARCHAR(MAX)= ERROR_MESSAGE()
    RAISERROR (@msg, 16, 1)
    -- error is 50000, rather than the original error value
END CATCH

-- SQL 2012 with THROW
BEGIN TRY
    DECLARE @Number int = 1 / 0;
END TRY
BEGIN CATCH
    -- Log the error info, then re-throw it
    INSERT INTO ErrorLog VALUES(GETDATE(), ERROR_SEVERITY(), ERROR_MESSAGE());
    THROW;
END CATCH

SELECT * FROM ErrorLog;

```

In Review

- OVER Clause Enhancements
 - ROWS/RANGE PRECEDING, FOLLOWING, UNBOUNDED
- IIF and CHOOSE
- OFFSET / FETCH
- FORMAT, CONCAT
- SEQUENCE
- sp_describe_first_result_set
- New Date and Time Functions
- Conversion Functions
 - PARSE, TRY_PARSE, TRY_CONVERT
- THROW exception

More Information

- Follow me on Twitter
 - @SqlEmt
- Database Health Project
 - <http://DatabaseHealth.SteveStedman.com>
- Visit my website
 - <http://stevestedman.com/>
- Send me an email:
 - Steve@SteveStedman.com
- Download Slides and Sample TSQL
 - <http://stevestedman.com/speaking>
- Next Presentation: Unleashing Common Table Expressions at 2:30pm in Room 4
