

# Time Functions, Logical Functions, and User Defined Functions



Presented by Steve Stedman  
and Aaron Buma

# Welcome

- Welcome to all those joining us remotely.
  - For any questions via Google On Air Broadcasts, we will address most of these at the end of the training.
- Training provided by Emergency Reporting
  - <http://EmergencyReporting.com>
- Slides and sample code are available at:
  - <http://SteveStedman.com>

# Welcome Viewers From

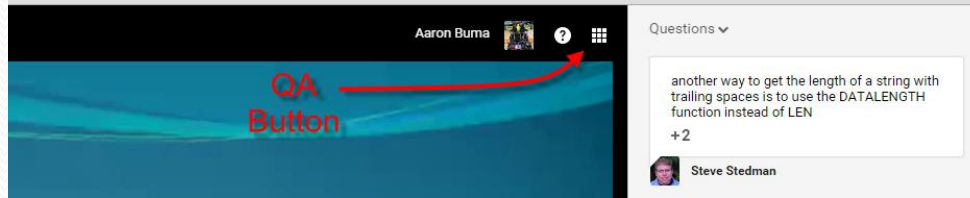
- Bellingham, WA

# Live Broadcast

- Using Google On Air Broadcasts
  - There is about a 40 to 50 second delay from live to what you see.
  - We are still learning how to properly use Google On Air Broadcasts. Please be patient.
  - Session will be available on my YouTube Channel about an hour after the presentation ends.
    - <http://SteveStedman.com/YouTube>

# Questions

- We will have time for questions at the end of the session.
- Q&A available via Google On Air Hangout panel. Click the 3x3 grid icon near the top right, then select Q&A to see what people are asking, or to ask your own question.



- When you ask a question, it shows up for us about 40 to 50 seconds delayed.

# Agenda

- Date and Time Functions
- Logical Functions
- User Defined Functions

# Date and Time Functions



Presented by Steve Stedman

# Date/Time Functions

- GETDATE()
- DATEPART()
- DATEADD()
- DATEDIFF()
- \_\_\_\_\_FROMPARTS (2012)
- EOMONTH (2012)



# GETDATE

- Returns the current database system timestamp as a **datetime** value without the database time zone offset.

```
SELECT GETDATE();
```

# DATEPART()

- Returns an integer that represents the specified *datepart* of the specified *date*.

```
SELECT DATEPART(year, GETDATE());
```

- Date can be **time**, **date**, **smalldatetime**, **datetime**, **datetime2**, or **datetimeoffset**.

# DATEADD()

- Returns a specified *date* with the specified *number* interval (signed integer) added to a specified *datepart* of that *date*.

```
SELECT DATEADD(month, 1, GETDATE());
```

# DATEDIFF()

- Returns the count (signed integer) of the specified *datepart* boundaries crossed between the specified *startdate* and *enddate*.

```
SELECT DATEDIFF(year, '1/1/2015', GETDATE());
```

```
SELECT DATEDIFF(quarter, '1/1/2015', GETDATE());
```

```
SELECT DATEDIFF(month, '1/1/2015', GETDATE());
```

# DATEFROMPARTS (2012)

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 (2012)

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 (2012)

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 (2012)

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 (2012)

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

- Same idea as the others....

# DATETIMEOFFSETFROMPARTS (2012)

- 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 (2012)

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*.

# Demo

Date and Time Functions

# Logical Functions



Presented by Steve Stedman

# Logical Functions

- CASE
- IIF (2012)
- CHOOSE (2012)
- COALESCE

# CASE

- Evaluates a list of conditions and returns one of multiple possible result expressions.

```
CASE WHEN Revenue > AverageRevenue  
      THEN 'Better Than Average'  
      ELSE 'Not Better'  
      END AS Ranking
```

# CASE with multiple WHEN

```
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;
```



# IIF (2012)

- Returns one of two values, depending on whether the Boolean expression evaluates to true or false in SQL Server.
- This is similar to Excel or VB versions of IIF.

```
IIF (Revenue > AverageRevenue,  
    'Better Than Average',  
    'Not Better' ) AS Ranking
```

- What does IIF stand for? Immediate IF? Inline IF?

# IIF Details

- Performance very similar between IIF and CASE
- IIF simplifies the code over using a CASE statement
- Can be nested up to 10 levels
- The true value and false value cannot both be NULL.

```
-- 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 (2012)

- 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;
```

# COALESCE

- Similar to CASE, or CHOOSE, but returns the first NON NULL value.
- Similar to ISNULL when using 2 parameters

`COALESCE([Parent], 0)`

- Will accommodate more than 2 fields.

`COALESCE([Parent], [AnotherField], 0)`

# Demo

Logical Functions

# User Defined Functions



Presented by Aaron Buma

# User Defined Functions

- Scalar
  - Multiple inputs, one value output
- Table Valued
  - Multiple inputs, row(s) as output
- Views
  - Referencing a query as a table



# Scalar Functions - Overview

- What it does:
  - Uses one (or many inputs)
  - To run one or more queries
  - And Return a single value
- What it doesn't:
  - Call non-deterministic functions (ie: `GetDate()`)
  - Insert, Update, Deletes to tables or views
  - Error handling
  - Get included in an execution plan (!Important)

# Functions in Views

- Deterministic Functions ex: LEFT()
  - You can index the view
- Non-Deterministic Functions ex: GETDATE()
  - You cannot index the view, because the data is always changing
  - For UDF's you need SCHEMABINDING at the function and view objects

# Table Value Function- Overview

- What it does:
  - IN: Zero or Many inputs, OUT: a result set
  - Single-Statement or Multi-Statement formatting
- What it doesn't:
  - Call non-deterministic functions (ie: GetDate())
  - Insert, Update, Deletes to tables or views
  - Error handling
  - Multi-statement are not included in the execution plan

# Demo

User Defined Functions

# Any Questions?

- Date and Time Functions
- Logical Functions
- User Defined Functions

# Tune in next week

- Thursday 3/26 at 9:00am pacific time
- Back to Basics – JOINS  
INNER, LEFT OUTER, RIGHT OUTER, SEMI JOIN,  
ANTI SEMI JOIN, LEFT OUTER with exclusion,  
RIGHT OUTER with exclusion, FULL OUTER, CROSS  
JOIN, FULL OUTER JOIN with exclusion, LATERAL  
JOINS, CROSS APPLY, combinations.

# For More Information

- Visit <http://EmergencyReporting.com> to find out more about Emergency Reporting.
- Aaron on the web
  - <http://AaronBuma.com>
  - Twitter: @AaronDBuma
- Steve on the web
  - <http://SteveStedman.com>
  - twitter: @SqlEmt