Trigger: Example

```
CREATE TRIGGER trg_employee
   ON  employee
   AFTER INSERT
AS
BEGIN
     -- SET NOCOUNT ON added to prevent extra
result sets from
     -- interfering with SELECT statements.
SET NOCOUNT ON;

     -- Insert statements for trigger here
INSERT INTO [employeeaudit](
       [emp_id]
      ,[first_name]
      ,[last_name]
      ,[mgr_id]
      ,[phone_number]
      ,[hire_date]
      ,[job_id]
      ,[deptno]
       ,machinename
      ,serverName
      ,[InsertTime])

-- continue….
```

```
-- …. Continued….

SELECT [emp_id]
  ,[first_name]
  ,[last_name]
  ,[mgr_id]
  ,[phone_number]
  ,[hire_date]
  ,[job_id]
  ,[deptno]
  ,CAST( SERVERPROPERTY('MachineName') AS VARCHAR(50))
  ,CAST( SERVERPROPERTY('ServerName') AS VARCHAR(50))
  ,GETDATE()
FROM INSERTED;


END
GO
```

Triggers: Benefits

Following are the benefits of triggers:

- Provides alternative method for implementing referential integrity constraint
- Restricts/Controls updates on the database objects
- can be used to publish information about database events (startup & shutdown) to subscribers

Triggers

A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database server. DML triggers execute when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view

Syntax:

```
CREATE [ OR ALTER ] TRIGGER schema_name.trigger_name ON {
table | view }
FOR | AFTER | INSTEAD OF
[ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ]
AS
sql_statement;
```

User Defined Functions (UDF): Benefits

Modular Programming

Functions are stored in db and can be call any number of times in your program

Faster Execution

Like stored procedures, all functions are compiled in the database and maintains the execution plan for subsequent calls

Reduce network traffic

A complex constraint that cannot be expressed as a single scalar expression can be written in a function and can be invoked from WHERE clause

Multi-Statement Table-Valued Function: Example

```
CREATE FUNCTION fn_GetEmployeeInfo(@DeptID
INTEGER)
RETURNS @retGetEmployeeInfo TABLE
(
    EmployeeID int primary key NOT NULL,
    FirstName nvarchar(255) NOT NULL,
    LastName nvarchar(255) NOT NULL,
    Deptno INT NOT NULL
)
AS
BEGIN
INSERT @retGetEmployeeInfo
SELECT [emp_id]
      ,[first_name]
      ,[last_name]
      ,[deptno]
FROM [empDB].[dbo].[employee]
WHERE deptno = @DeptID

RETURN
END;
GO
```

```
Execute:


USE [empDB]

GO


SELECT EmployeeID, FirstName, LastName,

Deptno

FROM fn_GetEmployeeInfo(10);
```

UDF- Multi-Statement Table: Valued Function

User-defined functions that return a table data type can be powerful alternatives to views

A table-valued user-defined function can be used where table or view expressions are allowed in Transact-SQL queries views are limited to a single SELECT statement, user-defined functions can contain additional statements

In a table-valued user-defined function:

- The RETURNS clause defines the format of the table. The scope of the local return variable name is local within the function
- The Transact-SQL statements in the function body build and insert rows into the return variable defined by the RETURNS clause
- The rows inserted into the variable are returned as the tabular output of the function

```
CREATE [OR ALTER] FUNCTION schema_name. function_name ( @parameter_name  AS parameter_data_type )

RETURNS @return_variable TABLE <table_type_definition>

 AS

BEGIN

<SQL STATEMENTS/COMPUTATION>

RETURN;

END;
```

Inline Table-Valued Function: Example

```
CREATE FUNCTION fn_GetEmployeeInfo (@empid
int)
RETURNS TABLE
AS
RETURN
(
SELECT[emp_id]
        ,[first_name]
        ,[last_name]
        ,[mgr_id]
        ,[phone_number]
        ,[hire_date]
        ,[job_id]
        ,[deptno]
FROM [empDB].[dbo].[employee]
WHERE emp_id = @empid
);

GO
```

```
Execute:

USE [empDB]

GO


select * from  fn_GetEmployeeInfo(1001)
```

UDF: Inline Table-Valued Function

Inline table valued functions are a subset of user-defined functions that return a table data type

Inline functions can be used to achieve the functionality of parameterized views

Inline table valued functions can be used to support parameters in the search conditions specified in the WHERE clause

Inline user-defined functions follow these rules:

- The RETURNS clause contains only the keyword **table**. Format need not be defined for return value
- There is no *function_body* delimited by BEGIN and END.
- The RETURN clause contains a single SELECT statement in parentheses.
- The table-valued function accepts only constants or **@local_variable** arguments

```
CREATE [OR ALTER] FUNCTION schema_name.function_name (@parameter_name  AS parameter_data_type )
RETURNS TABLE
AS
BEGIN
RETURN  <SQL STATEMENTS>;
END;
```

Scalar Function: Example

```
CREATE FUNCTION fn_getdeptno(@empid INT)
RETURNS INT
BEGIN

RETURN (SELECT DEPTNO
FROM EMPLOYEE
WHERE EMP_ID = @empid)

END
```

```
Execute:


USE [empDB]
GO


select * from employee
where deptno = dbo.fn_getdeptno(1001);
```

UDF: Scalar Function

- Scalar functions return a single data value of the type defined in the RETURNS clause
- No function body is required for an inline Scalar function
- When multi statements need to be provided within BEGIN-END block

Syntax:

```
CREATE [OR ALTER] FUNCTION schema_name.
function_name(@parameter_name  AS parameter_data_type)
RETURNS return_data_type
 AS
BEGIN
            <SQL STATEMENTS/COMPUTATION>
            RETURN scalar_value;
END;
```

User Defined Functions (UDF)

Like functions in programming languages, User Defined Functions are compact pieces of Transact SQL code, which can accept parameters, perform complex calculation and return either a value, or a table

Scalar Valued Function

A scalar value function is used to return the single value only like Integers or may be timestamp

Table Valued Function

A table valued function is used for any number of the row set values. It is useful in the case of the returning multiple rows set at the same time

All procedures by default are compiled in the database the first time it is executed and maintains the execution plan for subsequent calls

Stored Procedure: Example

```
Create  PROCEDURE sp_GetEmployeeInfo(@empid INT)
AS
BEGIN
SELECT[emp_id]
      ,[first_name]
      ,[last_name]
      ,[mgr_id]
      ,[phone_number]
      ,[hire_date]
      ,[job_id]
      ,[deptno]
FROM [empDB].[dbo].[employee]
WHERE emp_id = @empid
END;
```

```
Execute:


USE [empDB]
GO


DECLARE @return_value int


EXEC @return_value = [dbo].[GetEmployeeInfo]
@empid = 1001


SELECT 'Return Value' = @return_value
```

Stored Procedure: Benefits

Reduced server/client network traffic

All commands in a procedure are executed as a single batch of code

Stronger security

The EXECUTE AS can be specified to enable impersonating another user to perform certain db tasks without providing direct permission to the procedure

Reuse of code

The code (procedure) can be called from other procedures

Easier maintenance

Changes need to be done within stored procedure which handles all db operations and no changes are required in client applications

Improved performance

All procedures by default are compiled in the database the first time it is executed and maintains the execution plan for subsequent calls

Stored Procedure: Syntax

Syntax:

```
CREATE [OR ALTER] PROCEDURE
schema_name.procedure_name (
@parameter_name  AS parameter_data_type
[OUT])
AS
BEGIN
            <SQL STATEMENTS/COMPUTATION>
END;
```

Note: Use ALTER instead of CREATE to make changes in the procedure

Stored Procedure

- A stored procedure is a set of sql statements with a name, that has been created and stored in the database
- Stored Procedure can be defined as the set of logical group of SQL statements which are grouped to perform a specific task
- Like other programming constructs they can:
  - ❖ Accept input parameters
  - ❖ Return multiple values (out parameters)
  - ❖ Contain programming statements
  - ❖ Can call other stored procedures/functions
  - ❖ Returns status to indicate success or failure

Date & Time Functions: Example

```
Example:
SELECT
SYSDATETIME() AS 'SYSDATETIME', CURRENT_TIMESTAMP AS 'TIMESTAMP',
DATEPART(year,'12-DEC-2017' ) AS 'DATEPART',
DATEDIFF (mm , '12/31/2015' , '10/23/2016' ) as 'DATEDIFF', DATEADD
(mm , 2 , '12/31/2015' ) AS 'DATEADD'

Result:
```

SYSDATETIME	TIMESTAMP	DATEPART	DATEDIFF	DATEADD
2017-12-24 09:27:23.1501314	2017-12-24 09:27:23.150	2017	10	2016-02-29 00:00:00.000

Date & Time Functions

Date and time functions are scalar functions that perform an operation on a date and time input value and returns either a string, numeric, or date and time value

Function	Syntax	Description
SYSDATETIME	SYSDATETIME()	Returns a **datetime2(7)** value that contains the date and time
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	Returns a **datetime** value that contains the date and time
DATEPART	DATEPART (*datepart* , *date*)	Returns an integer that represents the specified *datepart* of the specified *date*
DAY/MONTH/YEAR	DAY(date) / MONTH (date) / YEAR(date)	Returns an integer that represents the day/month/year part of the specified *date*
DATEDIFF	DATEDIFF (*datepart* , *startdate* , *enddate*)	Returns the number of date or time *datepart* boundaries that are crossed between two specified dates
DATEADD	DATEADD (*datepart* , *number* , *date*)	Returns a new **datetime** value by adding an interval to the specified *datepart* of the specified *date*

String Functions: Examples

Example:

```
SELECT REPLACE('abcdefghicde','cde','xxx') as
'REPLACE',  SUBSTRING('abcdef', 2, 3) AS SUBSTR,  LEFT('abcdefg',2) AS
'LEFT', RIGHT('abcdefg',2) AS 'RIGHT'
```

REPLACE	SUBSTR	LEFT	RIGHT
abxxxfghixxx	bcd	ab	fg

```
DECLARE @d DATETIME = '10/01/2011';
SELECT FORMAT ( @d, 'd', 'en-US' ) AS 'US English'
      ,FORMAT ( @d, 'd', 'en-gb' ) AS 'Great Britain English'
      ,FORMAT ( @d, 'd', 'de-de' ) AS 'German'
```

US English	Great Britain English	German
10/1/2011	01/10/2011	01.10.2011

String Functions Cont'd...

Function	Syntax	Description
REPLACE	REPLACE(string, pattern, replacement)	Replaces all occurrences of a specified string value with another string value
SUBSTRING	SUBSTRING(expr, start, length)	Returns part of a character, text
FORMAT	FORMAT value, format [, culture])	Returns a value formatted with the specified format and optional culture
LEFT	LEFT(expr, int)	Returns the LEFT part of a character string with the specified number of characters
RIGHT	RIGHT(expr, int)	Returns the RIGHT part of a character string with the specified number of characters
REVERSE	REVERSE(string)	Returns the reverse order of a string value
UPPER	UPPER(string)	Returns a character expression with lowercase character data converted to uppercase
LOWER	LOWER(string)	Returns a character expression after converting uppercase character data to lowercase
LEN	LEN(string)	Returns the number of characters of the specified string expression, excluding trailing blanks

String Functions

- The below scalar functions perform an operation on a string input value and return a string or numeric value
- When string functions are passed arguments that are not string values, the input type is implicitly converted to a text data type

Function	Syntax	Description
LTRIM	LTRIM (char_expr)	Returns a character expr after it removes leading blanks
RTRIM	RTRIM (char_expr)	Returns a character string after truncating all trailing blanks
CHAR	CHAR (int_expr)	Converts an **int** ASCII code to a character
CHARINDEX	CHARINDEX (exprToFind , exprToSearch [, start_location])	Searches an expression for another expression and returns its starting position if found
STR	STR(expression)	Returns character data converted from numeric data
CONCAT	CONCAT(str1, str2, str3..)	Returns a string that is the result of concatenating two or more string values

Aggregate Functions: Examples

```
Example:
-- Get Avg, Min, Max, Sum of Salary from emp table
SELECT  AVG(SAL) AS AVGSAL, MIN(SAL) AS MINSAL, MAX(SAL) AS MAXSAL, SUM(SAL)
AS SUMSAL FROM EMPLOYEE

-- Get emp count for each dept
SELECT DEPTNO, COUNT(*) FROM EMPLOYEE
GROUP BY deptno



-- deptno wise avg, min, max, sum salary, with sum of sal > 3000
SELECT DEPTNO, AVG(SAL) AS AVGSAL, MIN(SAL) AS MINSAL, MAX(SAL) AS MAXSAL,
SUM(SAL) AS SUMSAL FROM EMPLOYEE
GROUP BY DEPTNO
HAVING SUM(SAL) > 3000
```

Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single value. With the exception of the COUNT aggregate function, all other aggregate functions ignore NULL values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement

Function	Syntax	Description
AVG	AVG (expr)	Returns the average of the values in a group. Null values are ignored.
MIN	MIN (expr)	Returns the minimum value in the expr.
SUM	SUM(expr)	Returns the sum of all the values, or only the DISTINCT values, in the expr. SUM can be used with numeric columns only. Null values are ignored.
COUNT	COUNT()	Returns the number of items in a group. COUNT works like the COUNT_BIG function. The only difference between the two functions is their return values. COUNT always returns an **int** data type value. COUNT_BIG always returns a **bigint** data type value.
MAX	MAX(expr)	Returns the maximum value in the expr.

Math Functions: Examples

```
SELECT ABS(-10) as 'ABS', RAND() AS 'RAND', EXP(4) AS 'EXP', FLOOR(4.66) AS
'FLOOR', CEILING(4.33) AS 'CEILING', SIGN(-8) AS 'SIGN', SQRT(2.56) AS
'SQRT', SQUARE(4) AS 'SQUARE', POWER(2, 3 ) AS 'POWER', ROUND (2.5677888 ,
2) AS 'ROUND'

Result:
```

ABS	RAND	EXP	FLOOR	CEILING	SIGN	SQRT	SQUARE	POWER	ROUND
10	0.117169346	54.59815003	4	5	-1	1.6	16	8	2.57

Math Functions

Math functions can be used to calculate business and Engineering calculations

Function	Syntax	Description
ABS	ABS (num_expr)	Returns absolute (positive) value of the specified numeric expr
RAND	RAND (seed)	Returns a pseudo-random float value from 0 to 1 (exclusive). Seed is optional value
EXP	EXP (float_expr)	Returns the exponential value
ROUND	ROUND (num_expr , length)	Returns a numeric value, rounded to the specified length or precision
FLOOR	FLOOR(num_expr)	Returns the largest integer less than or equal to the specified numeric expr
SIGN	SIGN (num_expr)	Returns the positive (+1), zero (0), or negative (-1) sign of the specified expr
SQRT	SQRT (float_expr)	Returns the square root of the specified float value
CEILING	CEILING (num_expr)	Returns the smallest integer greater than, or equal to, the specified numeric expr
SQUARE	SQUARE (float_expr)	Returns the square of the specified float value
POWER	POWER (float_expr, y)	Returns the value of the specified expr to the specified power

Logical Functions: Examples

```
Example:

SELECT CHOOSE(2, 'A', 'B', 'C') - returns B

-- CONSIDERS INTEGER VALUE ONLY and  returns A
SELECT CHOOSE(1.4, 'A', 'B', 'C')

SELECT CHOOSE(1.7, 'A', 'B', 'C')
```

```
SELECT IIF(1 > 10, 'TRUE', 'FALSE')

-- returns FALSE
```

Logical Functions

Logical functions can be used to display one of several values based on a logical conditions

Function	Syntax	Description
CHOOSE	CHOOSE (index, val_1, val_2 [, val_n])	Return a specified index from a list of values. Return NULL, if index is 0 or greater than number of items
IIF	IIF (boolean_expr, true_value, false_value)	Return one of two values, based on whether the Boolean expr evaluates to true or false

Conversion Functions: Examples

```
Example:
SELECT CAST('10' AS INT) * 20 AS CAST_VER
      , CONVERT(int, '10') * 20 AS CONVERT_VER
```

Note: Both functions can be used interchangeably in most situations
CAST is ANSI-SQL complaint while CONVERT is not

```
SELECT TRY_CAST('A100' AS INT) * 20 AS TRYCAST_VER
, TRY_CONVERT(INT, 'X100') * 20 AS TRYCONVERT_VER
-- Displays result as NULL as conversion not possible


SELECT CAST('A100' AS INT) * 20 AS TRYCAST_VER
-- Throws error as conversion failed
```

Conversion Functions

The conversion built-in functions are used to convert a expr from one data type to another

Function	Syntax	Description
CAST	CAST(expr AS datatype)	Returns value of specified datatype on success Throws error on failure
CONVERT	CONVERT(datatype, expr)	
PARSE	PARSE(value AS datatype)	
TRY_CAST	TRY_CAST(expr AS datatype)	Returns value of specified datatype on success NULL on failure
TRY_CONVERT	TRY_CONVERT(datatype, expr)	
TRY_PARSE	TRY_PARSE(value AS datatype)	

Built-in Functions

MS SQL built-in functions take zero or more inputs and returns a value.

Built-in Functions

- Gets system related information
- Used for calculations
- Manipulate input data

Function Category	Description
Conversion Function	Functions that support data type casting and converting
Logical Functions	Scalar functions that perform logical operation
Math Function	Scalar functions perform a calculation, usually based on input values that are provided as arguments, and return a numeric value
Aggregate Functions	Aggregate functions perform a calculation on a set of values and return a single value. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement
String Functions	Scalar functions perform an operation on a string input value and return a string or numeric value
Date Functions	Functions that manipulate dates

Character String Data Type

Data Type	Description	Length
char(n)	Stores *n* characters	*n* bytes (where *n* is in the range of 1–8,000)
nchar(n)	Stores *n* Unicode characters	2*n* bytes (where *n* is in the range of 1–4,000)
varchar(n)	Stores approximately *n*characters	Actual string length +2 bytes (where *n* is in the range of 1–8,000)
varchar(max)	Stores up to 231–1 characters	Actual string length +2 bytes
nvarchar(n)	Stores approximately *n*characters	2*n*(actual string length) +2 bytes (where *n* is in the range of 1–4,000)
nvarchar(max)	Stores up to ((231–1)/2)–2 characters	2*n*(actual string length) +2 bytes

Numeric Data Type

Data Type	Description	Length
int	Stores integer values ranging from -2,147,483,648 to 2,147,483,647	4 bytes
tinyint	Stores integer values ranging from 0 to 255	1 byte
smallint	Stores integer values ranging from -32,768 to 32,767	2 bytes
bigint	Stores integer values ranging from -253 to 253-1	8 bytes
money	Stores monetary values ranging from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
smallmoney	Stores monetary values ranging from -214,748.3648 to 214,748.3647	4 bytes
decimal(p,s)	Stores decimal values of precision *p* and scale *s*. The maximum precision is 38 digits	5–17 bytes
numeric(p,s)	Functionally equivalent to decimal	5–17 bytes
float(n)	Stores floating point values with precision of 7 digits (when *n*=24) or 15 digits (when *n*=53)	4 bytes (when *n*=24) or 8 bytes (when n=53)
real	Functionally equivalent to float(24)	4 bytes

Date and Time Data Type

Data Type	Description	Length	Example
date	Stores dates between January 1, 0001, and December 31, 9999	3 bytes	2006-05-13
datetime	Stores dates and times between January 1, 1753, and December 31, 9999, with an accuracy of 3.33 milliseconds	8 bytes	2006-03-18 03:34:56.132
datetime2	Stores date and times between January 1, 0001, and December 31, 9999, with an accuracy of 100 nanoseconds	6–8 bytes	2006-02-11 09:42:16.1322121
datetimeoffset	Stores date and times with the same precision as datetime2 and also includes an offset from Universal Time Coordinated (UTC) (also known as Greenwich Mean Time)	8-10 bytes	2006-02-11 09:42:16.1322121 +5:00
smalldatetime	Stores dates and times between January 1, 1900, and June 6, 2079, with an accuracy of 1 minute (the seconds are always listed as ":00")	4 bytes	2006-03-18 03:34:33
time	Stores times with an accuracy of 100 nanoseconds	3–5 bytes	09:42:16.1322121

Binary Data Type

Data Type	Description	Length
bit	Stores a single bit of data	1byte per 8 bit columns in a table
binary(n)	Stores n bytes of binary data	n bytes (where n is in the range of 1–8,000)
varbinary(n)	Stores approximately n bytes of binary data	Actual length +2 bytes (where n is in the range of 1–8,000)
varbinary(max)	Stores up to 231–1 bytes of binary data	Actual length +2 bytes

Other Data Types

Data Type	Description	Length
cursor	Stores a reference to a cursor	N/A (cannot be used in a table)
sql_variant	May store any data type other than sql_variant, text, ntext, image, and timestamp	Up to 8,000 bytes
table	Stores a temporary table (such as a query result)	N/A (cannot be used in a table)
rowversion	Stores a value of the database time (a relative number that increments each time you insert or update data in a database. It is not related to calendar/clock time)	8 bytes
uniqueidentifier	Stores a globally unique identifier	2 bytes
xml	Stores formatted XML documents	Up to 2GB