Data Management Using Microsoft SQL Server

Session: 13

Programming Transact-SQL

Objectives

- Describe an overview of Transact-SQL programming
- Describe the Transact-SQL programming elements
- Describe program flow statements
- Describe various Transact-SQL functions
- Explain the procedure to create and alter user-defined functions (UDFs)
- Explain creation of windows with OVER
- Describe window functions

© *Aptech Ltd.*

Introduction

- Transact-SQL programming is:
  - a procedural language extension to SQL.
  - extended by adding the subroutines and programming structures similar to high-level languages.

- Transact-SQL programming also has rules and syntax that control and enable programming statements to work together.

- Users can control the flow of programs by using conditional statements such as `IF` and loops such as `WHILE`.

Transact-SQL Programming Elements 1-2

Transact-SQL programming elements enable to perform various operations that cannot be done in a single statement.

Users can group several Transact-SQL statements together by using one of the following ways:

Batches

- Is a collection of one or more Transact-SQL statements that are sent as one unit from an application to the server.

Stored Procedures

- Is a collection of Transact-SQL statements that are precompiled and predefined on the server.

Triggers

- Is a special type of stored procedure that is executed when the user performs an event such as an `INSERT`, `DELETE`, or `UPDATE` operation on a table.

Scripts

- Is a chain of Transact-SQL statements stored in a file that is used as input to the SSMS code editor or `sqlcmd` utility.

Transact-SQL Programming Elements 2-2

The following features enable users to work with Transact-SQL statements:

Variables

- Allows a user to store data that can be used as input in a Transact-SQL statement.

Control-of-flow

- Is used for including conditional constructs in Transact-SQL.

Error Handling

- Is a mechanism that is used for handling errors and provides information to the users about the error occurred.

Transact-SQL Batches 1-5

Is a group of one or more Transact-SQL statements sent to the server as one unit from an application for execution.

SQL Server compiles the batch SQL statements into a single executable unit, also called as an execution plan.

In the execution plan, the SQL statements are executed one by one.

A Transact-SQL batch statement should be terminated with a semicolon.

A compile error such as syntax error restricts the compilation of the execution plan.

Transact-SQL Batches 2-5

A run-time error such as a constraint violation or an arithmetic overflow has one of the following effects:

- Most of the run-time errors stop the current statement and the statements that follow in the batch.
- A specific run-time error such as a constraint violation stops only the existing statement and the remaining statements in the batch are executed.

SQL statements that execute before the run-time error is encountered are unaffected.

© Aptech Ltd.

Transact-SQL Batches 3-5

Following rules are applied to use batches:

- CREATE FUNCTION, CREATE DEFAULT, CREATE RULE, CREATE TRIGGER, CREATE PROCEDURE, CREATE VIEW, and CREATE SCHEMA statements cannot be jointly used with other statements in a batch.
- CREATE SQL statement starts the batch and all other statements that are inside the batch will be considered as a part of the CREATE statement definition.
- No changes are made in the table and the new columns reference the same batch.
- If the first statement in a batch has the EXECUTE statement, then, the EXECUTE keyword is not required.

© Aptech Ltd.

Transact-SQL Batches 4-5

- Following code snippet shows how to create a batch:

```
BEGIN TRANSACTION
GO
USE AdventureWorks2012;
GO
CREATE TABLE Company
(
Id_Num int IDENTITY(100, 5),
Company_Name nvarchar(100)
)
GO
INSERT Company (Company_Name)
VALUES (N'A Bike Store')
INSERT Company (Company_Name)
VALUES (N'Progressive Sports')
INSERT Company (Company_Name)
VALUES (N'Modular Cycle Systems')
INSERT Company (Company_Name)
VALUES (N'Advanced Bike Components')
```

Transact-SQL Batches 5-5

```
INSERT Company (Company_Name)
VALUES (N'Metropolitan Sports Supply')
INSERT Company (Company_Name)
VALUES (N'Aerobic Exercise Company')
INSERT Company (Company_Name)
VALUES (N'Associated Bikes')
INSERT Company (Company_Name)
VALUES (N'Exemplary Cycles')
GO
SELECT Id_Num, Company_Name
FROM dbo. Company
ORDER BY Company_Name ASC;
GO
COMMIT;
GO
```

© Aptech Ltd.

Transact-SQL Variables 1-8

Variables allow users to store data for using as input in a Transact-SQL statement.

Users can use variables in the `WHERE` clause, and write the logic to store the variables with the appropriate data.

SQL Server provides the `DECLARE`, `SET`, and `SELECT` statements to set and declare local variables.

Transact-SQL Variables 2-8

DECLARE: Variables are assigned values by using the SELECT or SET statement and are initialized with NULL values if the user has not provided a value at the time of the declaration.

Syntax:

```
DECLARE {{ @local_variable [AS] data_type } | [ = value ] }
```

where,

@local_variable: specifies the name of the variables and begins with @ sign.

data_type: specifies the data type. A variable cannot be of image, text, or ntext data type.

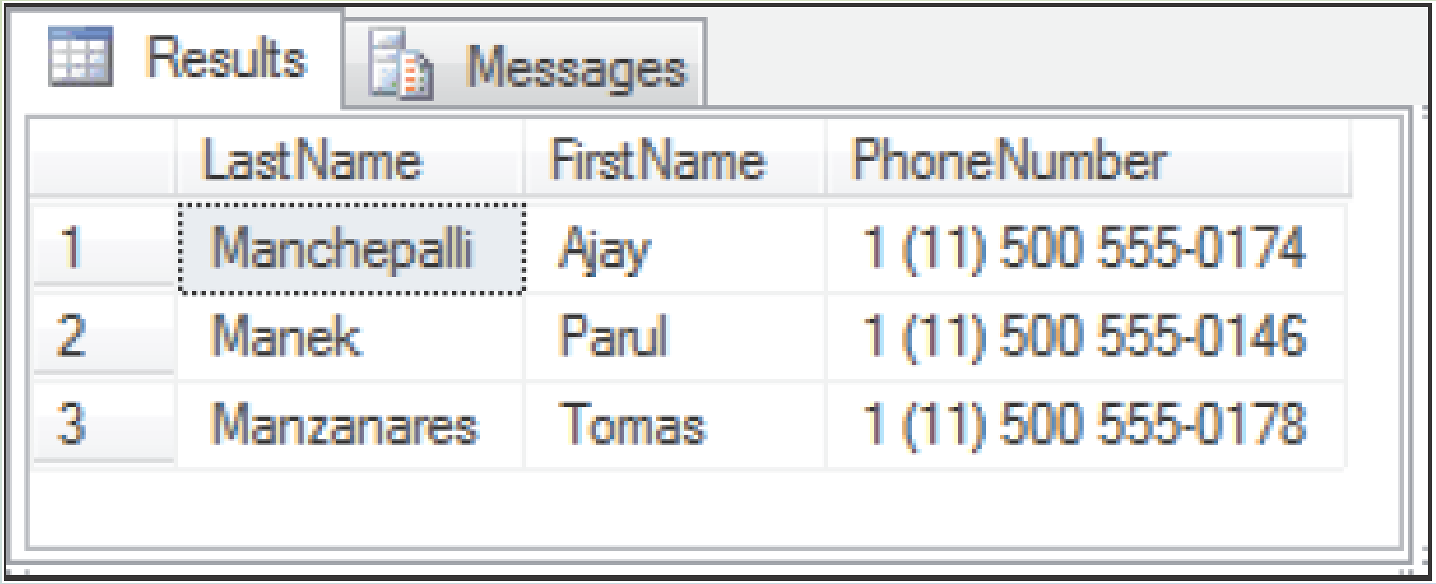
=value: Assigns an inline value to a variable. The value can be an expression or a constant value. The value should match with the variable declaration type or it should be implicitly converted to that type.

Transact-SQL Variables 3-8

- Following code snippet shows the use of a local variable to retrieve contact information for the last names starting with **Man**:

```
USE AdventureWorks2012;
GO
DECLARE @find varchar(30) = 'Man%';
SELECT p.LastName, p.FirstName, ph.PhoneNumber
FROM Person.Person AS p
JOIN Person.PersonPhone AS ph ON p.BusinessEntityID =
   ph.BusinessEntityID
WHERE LastName LIKE @find;
```

Output:



	LastName	FirstName	PhoneNumber
1	Manchepalli	Ajay	1 (11) 500 555-0174
2	Manek	Parul	1 (11) 500 555-0146
3	Manzanares	Tomas	1 (11) 500 555-0178

Transact-SQL Variables 4-8

SET: statement sets the local variable created by the DECLARE statement to the specified value.

Syntax:

```
SET
{ @local_variable = { expression}
}
|
{ @local_variable
{+= | -= | *= | /= | %= | &= | ^= | |= } expression
}
```

where,

`@local_variable`: specifies the name of the variables and begins with `@` sign.

`=`: Assigns the value on the right-hand side to the variable on the left-hand side.

`{= | += | -= | *= | /= | %= | &= | ^= | |= }`: specifies the compound assignment operators.

`expression`: specifies any valid expression which can even include a scalar subquery.

Transact-SQL Variables 5-8

- Following code snippet demonstrates the use of SET to assign a string value to a variable:

```
DECLARE @myvar char(20);
SET @myvar = 'This is a test';
```

SQL Server 2012

Transact-SQL Variables 6-8

SELECT: statement indicates that the specified local variable that was created using DECLARE should be set to the given expression.

Syntax:

```
SELECT { @local_variable { = | += | -= | *= | /= | %= | &= | ^= | |= }
   expression } [ ,...n ] [ ; ]
```

where,

@local_variable: specifies the name of the variables and begins with @ sign.

=: Assigns the value on the right-hand side to the variable on the left-hand side.

{= | += | -= | *= | /= | %= | &= | ^= | |= }: specifies the compound assignment operators.

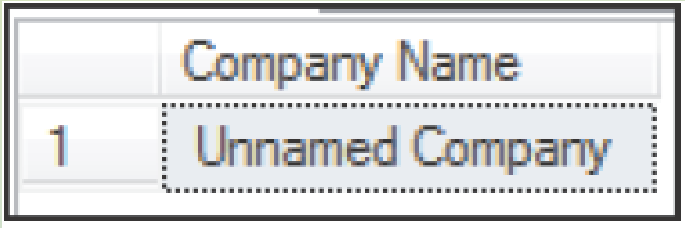
expression: specifies any valid expression which can even include a scalar subquery.

Transact-SQL Variables 7-8

- Following code snippet demonstrates the use of `SELECT` to return a single value:

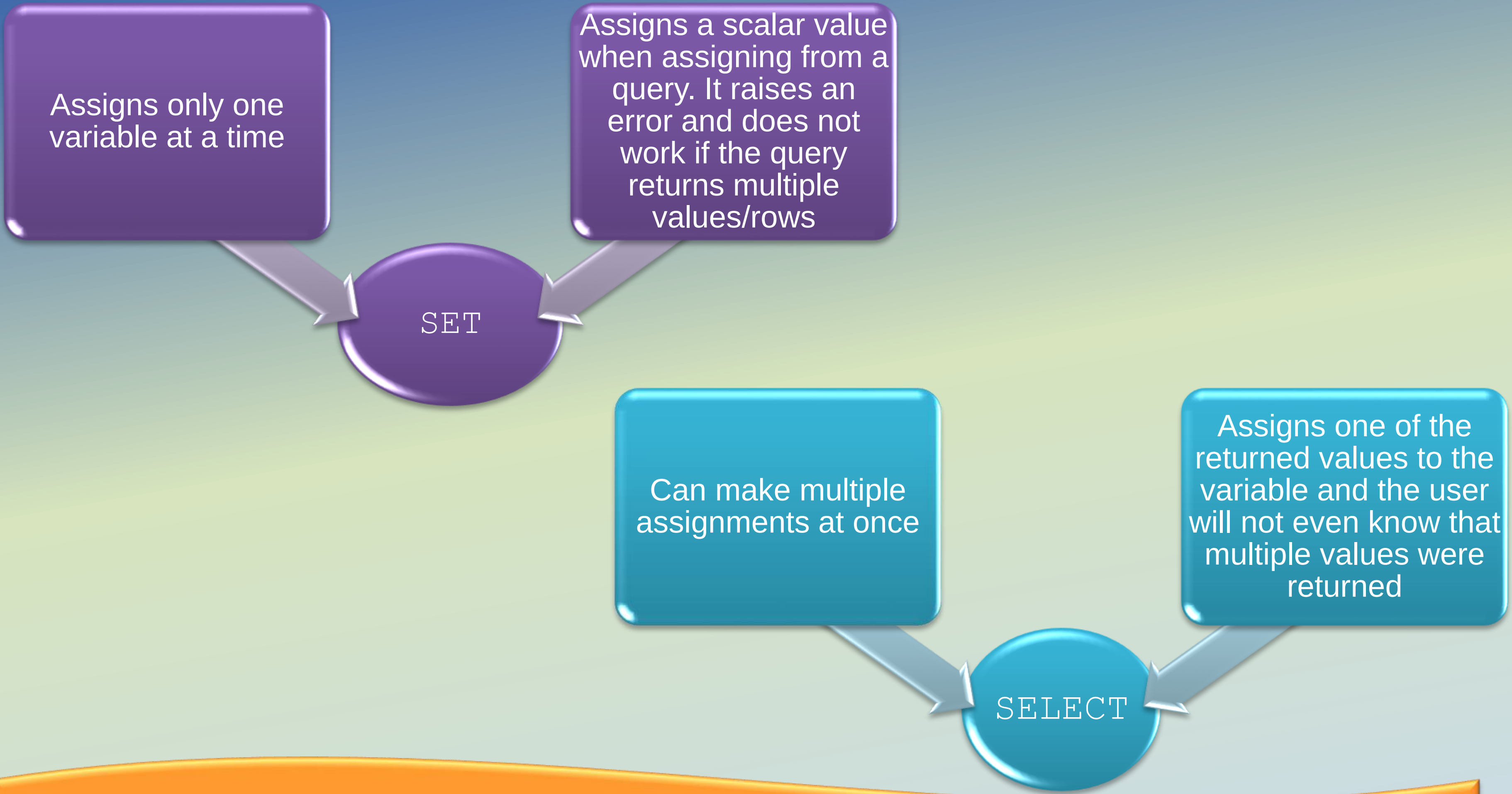
```
USE AdventureWorks2012 ;
GO
DECLARE @var1 nvarchar(30);
SELECT @var1 = 'Unnamed Company';
SELECT @var1 = Name
FROM Sales.Store
WHERE BusinessEntityID = 10;
SELECT @var1 AS 'Company Name';
```

Output:



	Company Name
1	Unnamed Company

Transact-SQL Variables 8-8



Assigns only one variable at a time

Assigns a scalar value when assigning from a query. It raises an error and does not work if the query returns multiple values/rows

SET

Can make multiple assignments at once

Assigns one of the returned values to the variable and the user will not even know that multiple values were returned

SELECT

To assign variables, it is recommended to use `SET @local_variable` instead of `SELECT @local_variable`

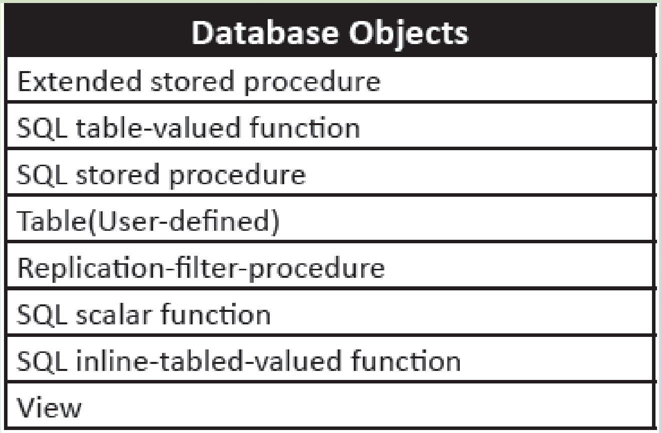
Synonyms 1-6

Are database objects that serve the following purposes:

- They offer another name for a different database object, also called as the base object, which may exist on a remote or local server.
- They present a layer of abstraction that guards a client application from the modifications made to the location and the name of the base object.

A synonym is a part of schema, and like other schema objects, the synonym name must be unique.

- Following table lists the database objects for which the users can create synonyms.



Database Objects
Extended stored procedure
SQL table-valued function
SQL stored procedure
Table(User-defined)
Replication-filter-procedure
SQL scalar function
SQL inline-tabled-valued function
View

© Aptech Ltd.

SQL Server 2012

Synonyms 2-6

- Synonyms and Schemas

Users want to create a synonym and have a default schema that is not owned by them.

They can qualify the synonym name with the schema name that they actually own.

- Granting Permissions on Synonyms

Only members of the roles `db_owner` or `db_ddladmin` or synonym owners are allowed to grant permissions on a synonym.

Users can deny, grant, or revoke all or any of the permissions on a synonym.

© *Aptech Ltd.*

Synonyms 3-6

- Working with Synonyms

Users can work with synonyms in SQL Server 2012 using either Transact-SQL or SSMS.

To create a synonym using SSMS, perform the following steps:

1) In Object Explorer, expand the database where you want to create a new synonym
2) Select the **Synonyms** folder, right-click it, and then, click **New Synonym**...
3) In the **New Synonym** dialog box, provide the following information:

Synonym name: is the new name for the object.

Synonym schema: is the new name for the schema object.

Server name: is the name of the server to be connected.

Database name: is the database name to connect the object.

Schema: is the schema that owns the object.

Synonyms 4-6

To create a synonym using Transact-SQL, perform the following steps:

1) Connect to the Database Engine.
2) Click **New Query** in the Standard bar.
3) Write the query to create the synonym in the query window.
4) Click **Execute** on the toolbar to complete creation of the synonym.

Synonyms 5-6

Syntax:

```
CREATE SYNONYM [ schema_name_1. ] synonym_name FOR <object>
<object> :: =
{

[ server_name.[ database_name ] . [ schema_name_2 ].| database_name . [
   schema_name_2 ].| schema_name_2. ] object_name
}
```

where,

`schema_name_1`: states that the schema in which the synonym is created.

`synonym_name`: specifies the new synonym name.

`server_name`: specifies the server name where the base object is located.

`database_name`: specifies the database name where the base object is located.

`schema_name_2`: specifies the schema name of the base object.

`object_name`: specifies the base object name, which is referenced by the synonym.

Synonyms 6-6

- Following code snippet creates a synonym from an existing table:

```
USE tempdb;
GO
CREATE SYNONYM MyAddressType
FOR AdventureWorks2012.Person.AddressType;
GO
```

Program Flow Statements 1-8

Different types of program flow statements and functions supported by Transact-SQL are as follows:

- Transact-SQL Control-of-Flow language

Determines the execution flow of Transact-SQL statements, statement blocks, user-defined functions, and stored procedures.

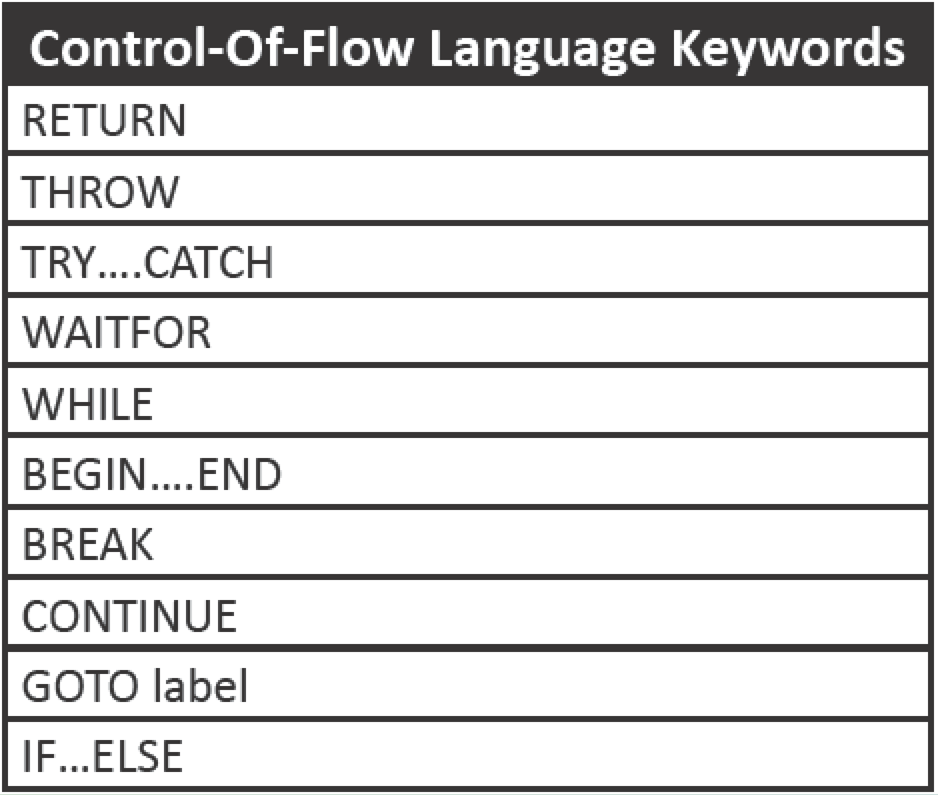
Transact-SQL statements are executed sequentially, in the order they occur.

Allow statements to be executed in a particular order, to be related to each other, and made interdependent using constructs similar to programming languages.

© Aptech Ltd.

Program Flow Statements 2-8

- Following table lists some of the Transact-SQL control-of-flow language keywords:



Control-Of-Flow Language Keywords
RETURN
THROW
TRY….CATCH
WAITFOR
WHILE
BEGIN….END
BREAK
CONTINUE
GOTO label
IF…ELSE

© *Aptech Ltd.*

SQL Server 2012

Program Flow Statements 3-8

BEGIN…END statements surround a series of Transact-SQL statements so that a group of Transact-SQL statements is executed.

Syntax:

```
BEGIN
{
 sql_statement | statement_block
}
END
```

where,

`{sql_statement| statement_block}`: Is any valid Transact-SQL statement that is defined using a statement block.

© Aptech Ltd.

Program Flow Statements 4-8

- Following code snippet shows the use of BEGIN and END statements:

```
USE AdventureWorks2012;
GO
BEGIN TRANSACTION;
GO
IF @@TRANCOUNT = 0
BEGIN
SELECT FirstName, MiddleName
FROM Person.Person WHERE LastName = 'Andy';
ROLLBACK TRANSACTION;
PRINT N'Rolling back the transaction two times would cause an error.';
END;
ROLLBACK TRANSACTION;
PRINT N'Rolled back the transaction.';
GO
```

© Aptech Ltd.

Program Flow Statements 5-8

IF…ELSE statement enforces a condition on the execution of a Transact-SQL statement.

Transact-SQL statement is followed with the IF keyword and the condition executes only if the condition is satisfied and returns TRUE.

ELSE keyword is an optional Transact-SQL statement that executes only when the IF condition is not satisfied and returns FALSE.

Syntax:

```
IF Boolean_expression

{ sql_statement | statement_block }
[ ELSE
{ sql_statement | statement_block } ]
```

where,

Boolean_expression: specifies the expression that returns TRUE or FALSE value

Program Flow Statements 6-8

`{sql_statement | statement_block}`: Is any valid Transact-SQL statement that is defined using a statement block.

- Following code snippet shows the use of `IF…ELSE` statements:

```
USE AdventureWorks2012
GO
DECLARE @ListPrice money;
SET @ListPrice = (SELECT MAX(p.ListPrice)
  FROM Production.Product AS p
  JOIN Production.ProductSubcategory AS s
  ON p.ProductSubcategoryID = s.ProductSubcategoryID
  WHERE s.[Name] = 'Mountain Bikes');
    PRINT @ListPrice
IF @ListPrice <3000
PRINT 'All the products in this category can be purchased for an amount
   less than 3000'
ELSE
PRINT 'The prices for some products in this category exceed 3000'
```

© Aptech Ltd.

Program Flow Statements 7-8

WHILE - statements specifies a condition for the repetitive execution of the statement block.

Statements are executed repetitively as long as the specified condition is true.

The execution of statements in the WHILE loop can be controlled by using the BREAK and CONTINUE keywords.

Syntax:

```
WHILE Boolean_expression
 { sql_statement | statement_block | BREAK | CONTINUE }
```

where,

Boolean_expression: specifies the expression that returns TRUE or FALSE value

{sql_statement| statement_block}: Is any valid Transact-SQL statement that is defined using a statement block.

BREAK: Results in an exit from the innermost WHILE loop.

CONTINUE: Results in the WHILE loop being restarted.

Program Flow Statements 8-8

- Following code snippet shows the use of `WHILE` statements:

```
DECLARE @flag int
SET @flag = 10
WHILE (@flag <=95)
  BEGIN
  IF @flag%2 =0
   PRINT @flag
   SET @flag = @flag + 1
   CONTINUE;
END
GO
```

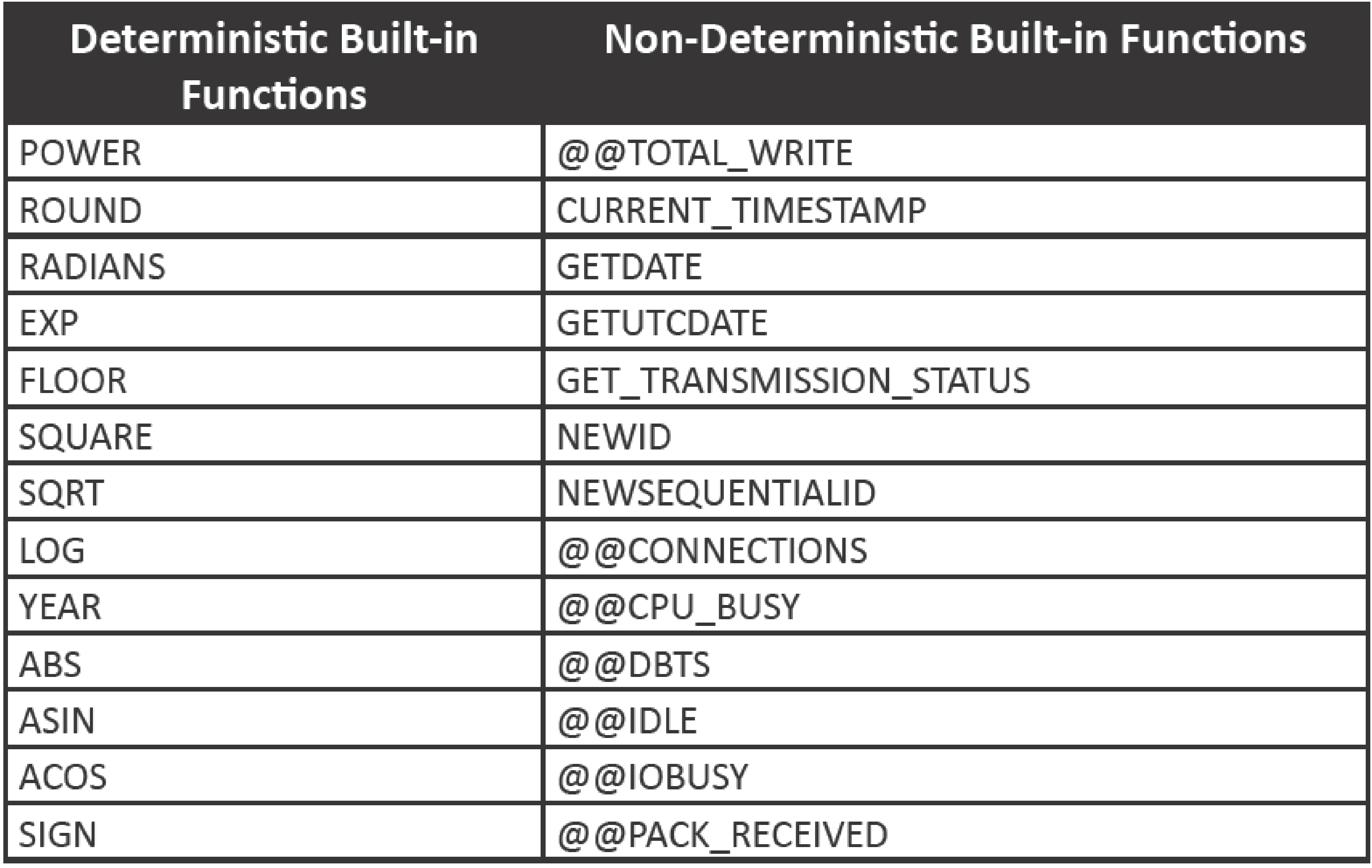
Transact-SQL Functions 1-5

Transact-SQL functions that are commonly used are as follows:

- **Deterministic and non-deterministic functions**
  - User-defined functions possess properties that define the capability of the SQL Server Database Engine.
  - Database engine is used to index the result of a function through either computed columns that the function calls or the indexed views that reference the functions.
  - Deterministic functions return the same result every time they are called with a definite set of input values and specify the same state of the database.
  - Non-deterministic functions return different results every time they are called with specified set of input values even though the database that is accessed remains the same.
  - Every built-in function is deterministic or non-deterministic depending on how the function is implemented by SQL Server.

Transact-SQL Functions 2-5

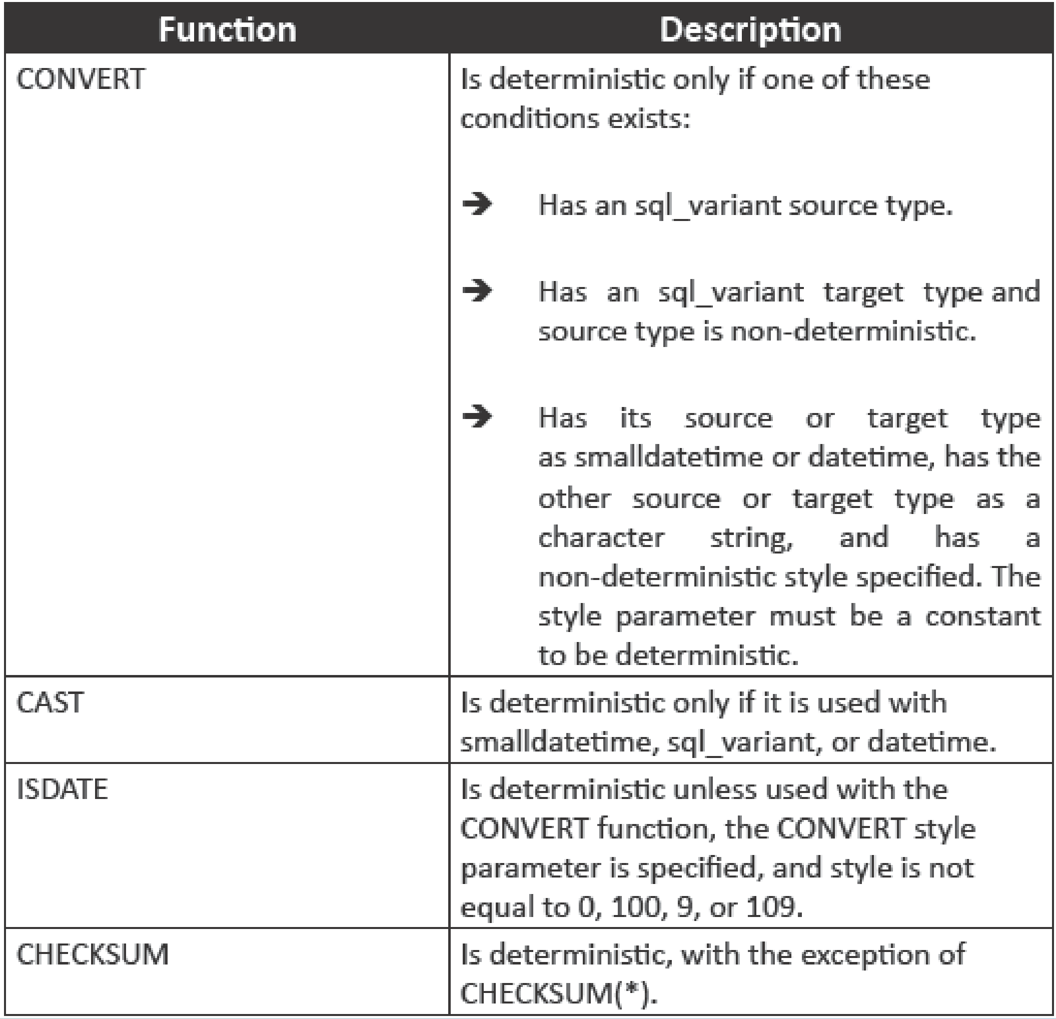
- Following table lists some deterministic and non-deterministic built-in functions:



Deterministic Built-in Functions	Non-Deterministic Built-in Functions
POWER	@@TOTAL_WRITE
ROUND	CURRENT_TIMESTAMP
RADIANS	GETDATE
EXP	GETUTCDATE
FLOOR	GET_TRANSMISSION_STATUS
SQUARE	NEWID
SQRT	NEWSEQUENTIALID
LOG	@@CONNECTIONS
YEAR	@@CPU_BUSY
ABS	@@DBTS
ASIN	@@IDLE
ACOS	@@IOBUSY
SIGN	@@PACK_RECEIVED

Transact-SQL Functions 3-5

- Following table lists some functions that are not always deterministic but you can use them in indexed views if they are given in a deterministic manner:



Function	Description
CONVERT	Is deterministic only if one of these conditions exists: ➔ Has an sql_variant source type. ➔ Has an sql_variant target type and source type is non-deterministic. ➔ Has its source or target type as smalldatetime or datetime, has the other source or target type as a character string, and has a non-deterministic style specified. The style parameter must be a constant to be deterministic.
CAST	Is deterministic only if it is used with smalldatetime, sql_variant, or datetime.
ISDATE	Is deterministic unless used with the CONVERT function, the CONVERT style parameter is specified, and style is not equal to 0, 100, 9, or 109.
CHECKSUM	Is deterministic, with the exception of CHECKSUM(*).

SQL Server 2012

Transact-SQL Functions 4-5

- **Calling Extended Stored Procedures from Functions**
  - Functions calling extended stored procedures are non-deterministic because the extended stored procedures may result in side effects on the database.
  - While executing an extended stored procedure from a user-defined function, the user cannot assure that it will return a consistent resultset.
  - Therefore, the user-defined functions that create side effects on the database are not recommended.
- **Scalar-Valued Functions**
  - A Scalar-Valued Function (SVF) always returns an `int`, `bit`, or `string` value.
  - Data type returned from and the input parameters of SVF can be of any data type except `text`, `ntext`, `image`, `cursor`, and `timestamp`.
  - An inline scalar function has a single statement and no function body.
  - A multi-statement scalar function encloses the function body in a `BEGIN...END` block.

Transact-SQL Functions 5-5

- **Table-Valued Functions**
  - Table-valued functions are user-defined functions that return a table.
  - Similar to an inline scalar function, an inline table-valued function has a single statement and no function body.
- Following code snippet shows the creation of a table-valued function:

```
USE AdventureWorks2012;
GO
IF OBJECT_ID (N'Sales.ufn_CustDates', N'IF') IS NOT NULL
  DROP FUNCTION Sales.ufn_ufn_CustDates;
GO
CREATE FUNCTION Sales.ufn_CustDates ()
RETURNS TABLE
AS
RETURN
(
SELECT A.CustomerID, B.DueDate, B.ShipDate
FROM Sales.Customer A
LEFT OUTER JOIN
Sales.SalesOrderHeader B
ON
A.CustomerID = B.CustomerID AND YEAR(B.DueDate)<2012
);
```

Altering User-defined Functions 1-3

Limitations and Restrictions

- ALTER FUNCTION does not allow the users to perform the following actions:
  - Modify a scalar-valued function to a table-valued function.
  - Modify an inline function to a multi-statement function.
  - Modify a Transact-SQL to a CLR function.

Permissions

- ALTER permission is required on the schema or the function.
- If the function specifies a user-defined type, then it requires the EXECUTE permission on the type.

Altering User-defined Functions 2-3

Modifying a User-defined function using SSMS

- Users can also modify user-defined functions using SSMS
- To modify the user-defined function using SSMS, perform the following steps:
  - Click the plus (+) symbol beside the database that contains the function to be modified.
  - Click the plus (+) symbol next to the Programmability folder.
  - Click the plus (+) symbol next to the folder, which contains the function to be modified.
  - Right-click the function to be modified and then, select Modify. The code for the function appears in a query editor window.
  - In the query editor window, make the required changes to the `ALTER FUNCTION` statement body.
  - Click Execute on the toolbar to execute the `ALTER FUNCTION` statement.

Altering User-defined Functions 3-3

Modifying a User-defined function using Transact-SQL

- To modify the user-defined function using Transact-SQL, perform the following steps:
  - In the Object Explorer, connect to the Database Engine instance.
  - On the Standard bar, click New Query.
  - Type the `ALTER FUNCTION` code in the Query Editor.
  - Click Execute on the toolbar to execute the `ALTER FUNCTION` statement.

➢ Following code snippet demonstrates modifying a table-valued function:

```
USE [AdventureWorks2012]
GO
ALTER FUNCTION [dbo].[ufnGetAccountingEndDate]()
RETURNS [datetime]
AS
BEGIN
RETURN DATEADD(millisecond, -2, CONVERT(datetime, '20040701', 112));
END;
```

Creation of Windows with OVER

A window function is a function that applies to a collection of rows.

The word 'window' is used to refer to the collection of rows that the function works on.

`OVER` clause is used to define a window within a query resultset.

SQL Server 2012

Windowing Components 1-3

- The three core components of creating windows with the OVER clause are as follows:

Partitioning - is a feature that limits the window of the recent calculation to only those rows from the resultset that contains the same values in the partition columns as in the existing row.

- Following code snippet demonstrates use of the PARTITION BY and OVER clauses with aggregate functions:

```
USE AdventureWorks2012;
GO
SELECT SalesOrderID, ProductID, OrderQty
,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS Total
,MAX(OrderQty) OVER(PARTITION BY SalesOrderID) AS MaxOrderQty
FROM Sales.SalesOrderDetail
WHERE ProductId IN(776, 773);
GO
```

© Aptech Ltd.

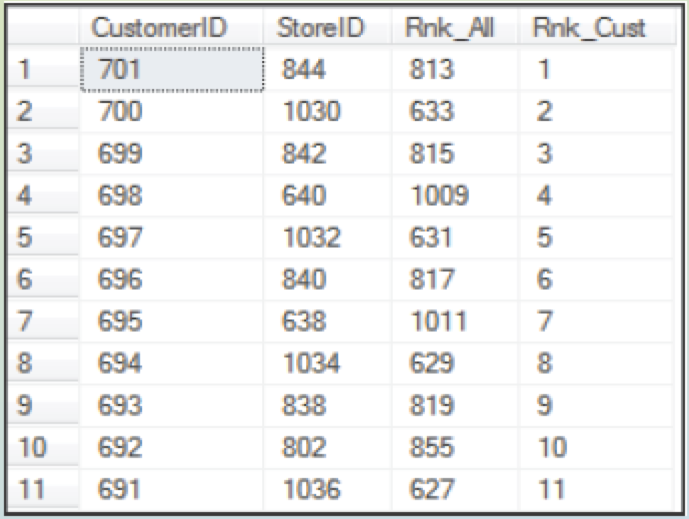
Windowing Components 2-3

Ordering - element defines the ordering for calculation in the partition. In SQL Server 2012, there is a support for the ordering element with aggregate functions.

- Following code snippet demonstrates an example of the ordering element:

```
SELECT CustomerID, StoreID,
RANK() OVER(ORDER BY StoreID DESC) AS Rnk_All,
RANK() OVER(PARTITION BY PersonID
ORDER BY CustomerID DESC) AS Rnk_Cust
FROM Sales.Customer;
```

Output:



	CustomerID	StoreID	Rnk_All	Rnk_Cust
1	701	844	813	1
2	700	1030	633	2
3	699	842	815	3
4	698	640	1009	4
5	697	1032	631	5
6	696	840	817	6
7	695	638	1011	7
8	694	1034	629	8
9	693	838	819	9
10	692	802	855	10
11	691	1036	627	11

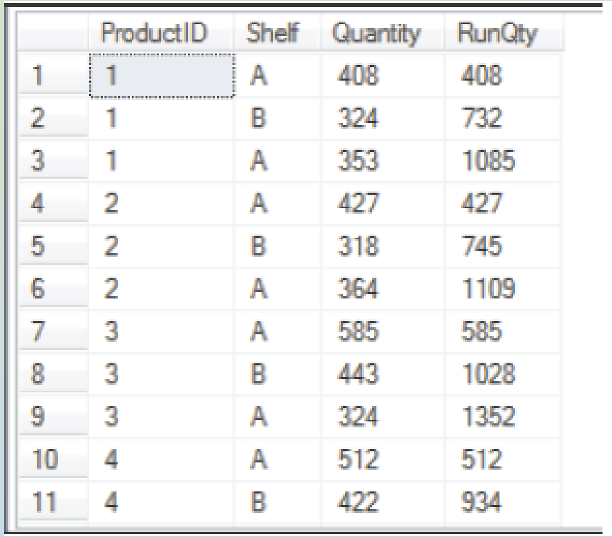
Windowing Components 3-3

Framing - is a feature that enables you to specify a further division of rows within a window partition. a frame is like a moving window over the data that starts and ends at specified positions and is defined using the ROW or RANGE subclauses.

- Following code snippet displays a query against the **`ProductInventory`**, calculating the running total quantity for each product and location:

```
SELECT ProductID, Shelf, Quantity,
SUM(Quantity) OVER(PARTITION BY ProductID
ORDER BY LocationID
ROWS BETWEEN UNBOUNDED PRECEDING
AND CURRENT ROW) AS RunQty
FROM Production.ProductInventory;
```

Output:



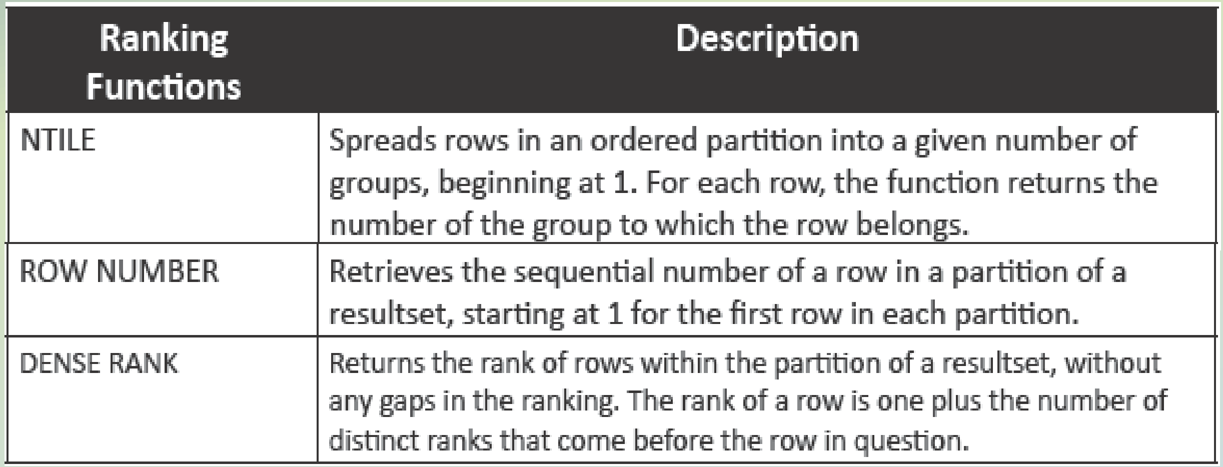
	ProductID	Shelf	Quantity	RunQty
1	1	A	408	408
2	1	B	324	732
3	1	A	353	1085
4	2	A	427	427
5	2	B	318	745
6	2	A	364	1109
7	3	A	585	585
8	3	B	443	1028
9	3	A	324	1352
10	4	A	512	512
11	4	B	422	934

Window Functions 1-9

- Some of the different types of window functions are as follows:

Ranking functions - return a rank value for each row in a partition. Based on the function that is used, many rows will return the same value as the other rows and are non-deterministic.

- Following table lists the various ranking functions:



Ranking Functions	Description
NTILE	Spreads rows in an ordered partition into a given number of groups, beginning at 1. For each row, the function returns the number of the group to which the row belongs.
ROW NUMBER	Retrieves the sequential number of a row in a partition of a resultset, starting at 1 for the first row in each partition.
DENSE RANK	Returns the rank of rows within the partition of a resultset, without any gaps in the ranking. The rank of a row is one plus the number of distinct ranks that come before the row in question.

Window Functions 2-9

- Following code snippet demonstrates the use of ranking functions:

```
USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName
,ROW_NUMBER() OVER (ORDER BY a.PostalCode) AS 'Row Number'
,NTILE(4) OVER (ORDER BY a.PostalCode) AS 'NTILE'
,s.SalesYTD, a.PostalCode
FROM Sales.SalesPerson AS s
INNER JOIN Person.Person AS p
ON s.BusinessEntityID = p.BusinessEntityID
INNER JOIN Person.Address AS a
ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
AND SalesYTD <> 0;
```

Window Functions 3-9

OFFSET functions - Different types of offset functions are as follows:

- SWITCHOFFSET - returns a DATETIMEOFFSET value that is modified from the stored time zone offset to a specific new time zone offset.

Syntax:

```
SWITCHOFFSET ( DATETIMEOFFSET, time_zone )
```

where,

DATETIMEOFFSET: is an expression that is resolved to a datetimeoffset(n) value.

time_zone: specifies the character string in the format [+|-]TZH:TZM or a signed integer (of minutes) which represents the time zone offset, and is assumed to be daylight-saving aware and adjusted.

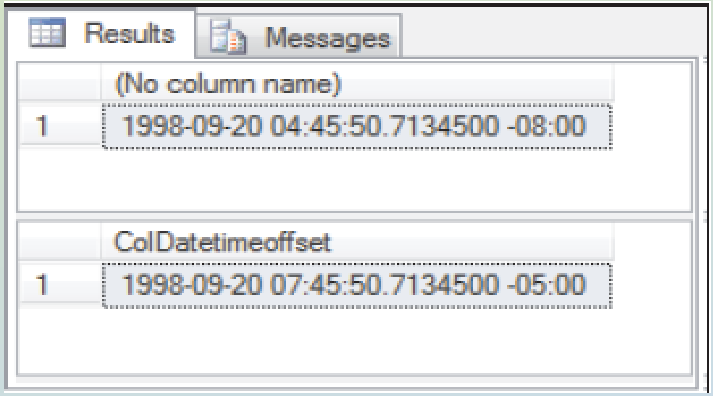
SQL Server 2012

Window Functions 4-9

- Following code snippet demonstrates the use of `SWITCHOFFSET` function:

```
CREATE TABLE Test
(
ColDatetimeoffset datetimeoffset
);
GO
INSERT INTO Test
VALUES ('1998-09-20 7:45:50.71345 -5:00');
GO
SELECT SWITCHOFFSET (ColDatetimeoffset, '-08:00')
FROM Test;
GO
--Returns: 1998-09-20 04:45:50.7134500 -08:00
SELECT ColDatetimeoffset
FROM Test;
```

Output:



	(No column name)
1	1998-09-20 04:45:50.7134500 -08:00

	ColDatetimeoffset
1	1998-09-20 07:45:50.7134500 -05:00

© Aptech Ltd.

SQL Server 2012

Window Functions 5-9

- DATETIMEOFFSETFROMPARTS – returns a datetimeoffset value for the specified date and time with specified precision and offset.

Syntax:

```
DATETIMEOFFSETFROMPARTS ( year, month, day, hour,
minute, seconds, fractions, hour_offset,
minute_offset, precision )
```

where,

year: specifies the integer expression for a year.

month: specifies the integer expression for a month.

day: specifies the integer expression for a day.

hour: specifies the integer expression for an hour.

minute: specifies the integer expression for a minute.

seconds: specifies the integer expression for a day.

fractions: specifies the integer expression for fractions.

hour_offset: specifies the integer expression for the hour portion of the time zone offset.

minute_offset: specifies the integer expression for the minute portion of the time zone offset.

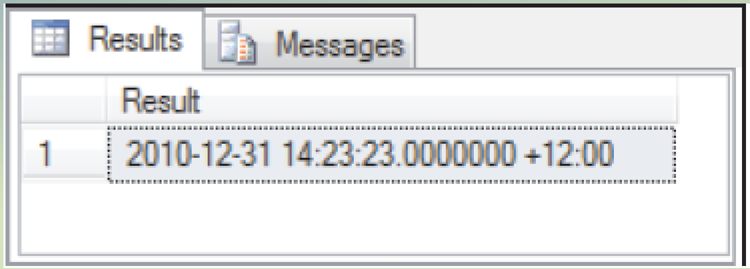
precision: specifies the integer literal precision of the datetimeoffset value to be returned.

Window Functions 6-9

- Following code snippet demonstrates the use of DATETIMEOFFSETFROMPARTS function:

```
SELECT DATETIMEOFFSETFROMPARTS ( 2010, 12, 31, 14, 23, 23, 0, 12, 0,
7 )
AS Result;
```

Output:



```
   Result
1  2010-12-31 14:23:23.0000000 +12:00
```

Window Functions 7-9

- SYSDATETIMEOFFSET – returns datetimeoffset(7) value which contains the date and time of the computer on which the instance of SQL Server is running.

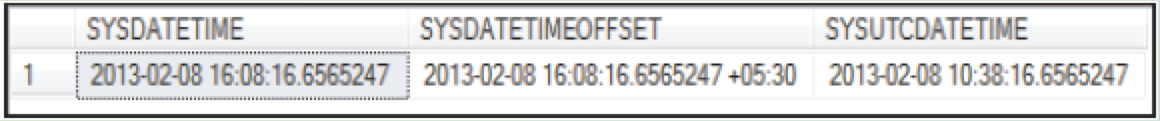
Syntax:

```
SYSDATETIMEOFFSET ()
```

- Following code snippet demonstrates the use of different formats used by the date and time functions:

```
SELECT SYSDATETIME() AS SYSDATETIME
,SYSDATETIMEOFFSET() AS SYSDATETIMEOFFSET
,SYSUTCDATETIME() AS SYSUTCDATETIME
```

Output:

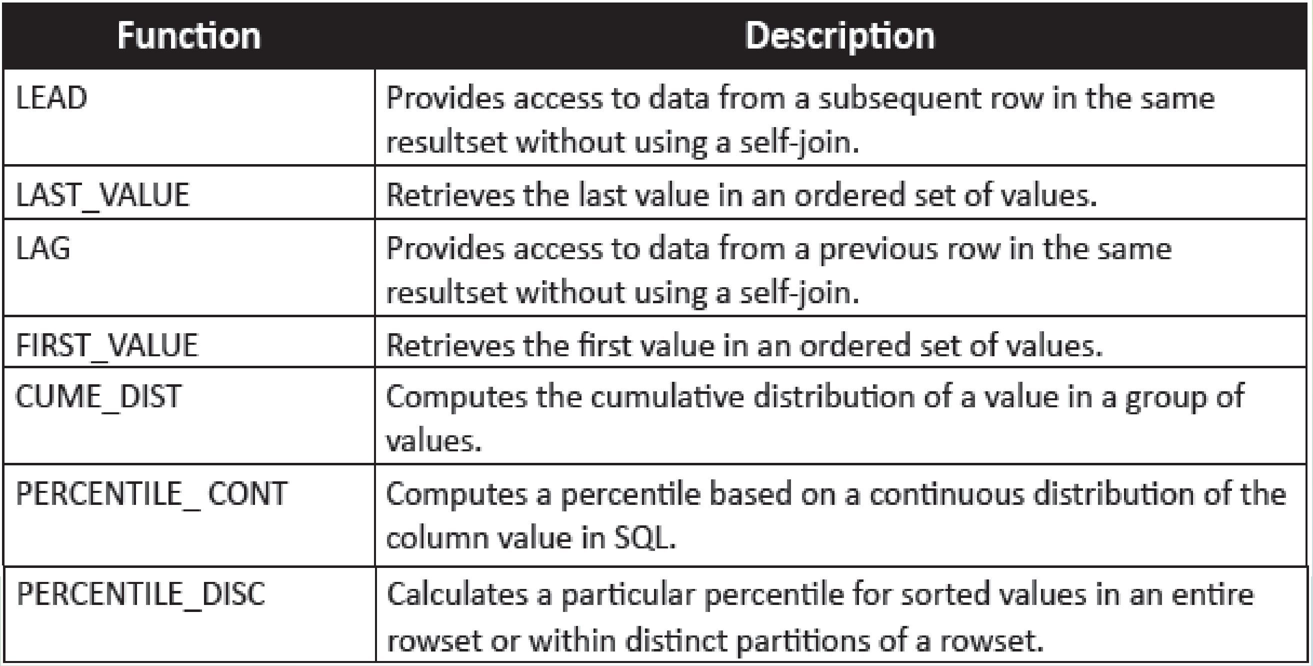


	SYSDATETIME	SYSDATETIMEOFFSET	SYSUTCDATETIME
1	2013-02-08 16:08:16.6565247	2013-02-08 16:08:16.6565247 +05:30	2013-02-08 10:38:16.6565247

Window Functions 8-9

Analytic functions - compute aggregate value based on a group of rows. Analytic functions compute running totals, moving averages, or top-N results within a group.

- Following table lists the various analytic functions:



Function	Description
LEAD	Provides access to data from a subsequent row in the same resultset without using a self-join.
LAST_VALUE	Retrieves the last value in an ordered set of values.
LAG	Provides access to data from a previous row in the same resultset without using a self-join.
FIRST_VALUE	Retrieves the first value in an ordered set of values.
CUME_DIST	Computes the cumulative distribution of a value in a group of values.
PERCENTILE_ CONT	Computes a percentile based on a continuous distribution of the column value in SQL.
PERCENTILE_DISC	Calculates a particular percentile for sorted values in an entire rowset or within distinct partitions of a rowset.

© Aptech Ltd.

Window Functions 9-9

- Following code snippet demonstrates the use of `LEAD()` function:

```
USE AdventureWorks2012;
GO
SELECT BusinessEntityID, YEAR(QuotaDate) AS QuotaYear, SalesQuota AS
   NewQuota,
LEAD(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS FutureQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 and YEAR(QuotaDate) IN ('2007','2008');
```

- Following code snippet demonstrates the use of `FIRST_VALUE()` function:

```
USE AdventureWorks2012;
GO
SELECT Name, ListPrice,
FIRST_VALUE(Name) OVER (ORDER BY ListPrice ASC) AS LessExpensive
FROM Production.Product
WHERE ProductSubcategoryID = 37
```

Summary

- Transact-SQL provides basic programming elements like variables, control-of-flow elements, conditional, and loop constructs.
- A batch is a collection of one or more Transact-SQL statements that are sent as one unit from an application to the server.
- Variables allow users to store data for using as input in other Transact-SQL statements.
- Synonyms provide a way to have an alias for a database object that may exist on a remote or local server.
- Deterministic functions each time return the same result every time they are called with a definite set of input values and specify the same state of the database.
- Non-deterministic functions return different results every time they are called with specified set of input values even though the database that is accessed remains the same.
- A window function is a function that applies to a collection of rows.