

Microsoft® SQL Server®

Notes for Professionals

Chapter 36: Common Table Expressions

Section 36.1: Generate a table of dates using CTE

```
DECLARE @StartDate datetime, @NumberOfDays int
SET @StartDate = '20180101'
SET @NumberOfDays = 18

WITH CTE_DatesTable
AS
SELECT CAST(@StartDate as date) AS [date]
UNION ALL
SELECT DATEADD(DAY, 1, [date])
FROM CTE_DatesTable
WHERE DATEADD(DAY, 1, [date]) <= DATEADD(DAY, @NumberOfDays-1, @StartDate)

SELECT [date] FROM CTE_DatesTable
OPTION (NOEXPLOSION ON)

This example returns a single column table of dates, starting with the date specified
returning the next @NumberOfDays worth of dates.
```

Section 36.2: Employee Hierarchy

```
Table Setup
CREATE TABLE dbo.Employees
(
  EmployeeID INT NOT NULL PRIMARY KEY,
  FirstName NVARCHAR(50) NOT NULL,
  LastName NVARCHAR(50) NOT NULL,
  ManagerID INT NULL
)
GO

INSERT INTO Employees VALUES (101, 'Ken', 'Sakuma', NULL)
INSERT INTO Employees VALUES (102, 'Fred', 'Steph', 101)
INSERT INTO Employees VALUES (103, 'John', 'Walker', 101)
INSERT INTO Employees VALUES (104, 'Zyhe', 'Klybo', 103)
INSERT INTO Employees VALUES (105, 'Sam', 'Jackson', 103)
INSERT INTO Employees VALUES (106, 'Peter', 'Smolko', 105)
INSERT INTO Employees VALUES (107, 'Gina', 'Wenkley', 105)
INSERT INTO Employees VALUES (108, 'George', 'Korringster', 105)
INSERT INTO Employees VALUES (110, 'Michael', 'Korringster', 108)

Common Table Expression
WITH cteReports (EmpID, FirstName, LastName, SupervisorID)
AS
SELECT EmployeeID, FirstName, LastName, ManagerID
FROM Employees
WHERE ManagerID IS NULL
UNION ALL
```

Chapter 59: Index

Section 59.1: Create Clustered Index

With a clustered index the leaf pages contain the actual table rows. Therefore, there can be only one clustered index.

```
CREATE TABLE Employees
(
  ID CHAR(90),
  FirstName NVARCHAR(3000),
  LastName NVARCHAR(3000),
  StartYear CHAR(900)
)
GO

CREATE CLUSTERED INDEX IX_Clustered
ON Employees (ID)
GO
```

Section 59.2: Drop index

```
DROP INDEX IX_NonClustered ON Employees
```

Section 59.3: Create Non-Clustered Index

Non-clustered indexes have a structure separate from the data rows. A non-clustered index contains the non-clustered index key values and each key value entry has a pointer to the data row that contains the key value. There can be maximum 999 non-clustered indexes on SQL Server 2008/2012.

Link for reference: <https://docs.microsoft.com/en-us/sql/indexes/ncli-43332.aspx>

```
CREATE TABLE Employees
(
  ID CHAR(90),
  FirstName NVARCHAR(3000),
  LastName NVARCHAR(3000),
  StartYear CHAR(900)
)
GO

CREATE NONCLUSTERED INDEX IX_NonClustered
ON Employees (StartYear)
GO
```

Section 59.4: Show index info

```
SP_HELPINDEX tablename
```

Section 59.5: Returns size and fragmentation indexes

```
sys.dm_db_index_physical_stats (
  @database_id = 'SQL', 0, @default
  , @object_id = NULL, 0, @default
  , @index_id = NULL, 0, @default
  , @index_id = NULL, 0, @default
)
```

Microsoft SQL Server® Notes for Professionals

Chapter 41: String Functions

Section 41.1: Quotename

Returns a Unicode string surrounded by delimiters to make it a valid SQL Server delimited identifier.

Parameters:

1. character string: A string of Unicode data, up to 128 characters (*sysname*). If an input string is longer than 128 characters, function returns null.
2. quote character, **Optional**: A single character to use as a delimiter. Can be a single quotation mark (') or ''', a left or right bracket ([], [, or],), or a double quotation mark ("). Any other value will return null. Default value is square brackets.

```
SELECT QUOTENAME('what's my name?') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what's my name?', '[') -- Returns [what's my name?]
```

Section 41.2: Replace

Returns a string (*varchar* or *nvarchar*) where all occurrences of a specified sub string is replaced with another sub string.

Parameters:

1. string expression: This is the string that would be searched. It can be a character or binary data type.
2. pattern: This is the sub string that would be replaced. It can be a character or binary data type. The pattern argument cannot be an empty string.
3. replacement: This is the sub string that would replace the pattern sub string. It can be a character or binary data.

```
SELECT REPLACE('this is my string', 'is', 'xx') -- Returns 'this xx my string'
```

Notes:

- If string expression is not of type *varchar*(*n*) or *nvarchar*(*n*), the *replace* function truncates the return value at 8000 chars.
- Return data type depends on input data types - returns *nvarchar* if one of the input values is *nvarchar*, or *varchar* otherwise.

Microsoft SQL Server® Notes for Professionals

200+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with Microsoft SQL Server	2
Section 1.1: INSERT / SELECT / UPDATE / DELETE: the basics of Data Manipulation Language	2
Section 1.2: SELECT all rows and columns from a table	6
Section 1.3: UPDATE Specific Row	6
Section 1.4: DELETE All Rows	7
Section 1.5: Comments in code	7
Section 1.6: PRINT	8
Section 1.7: Select rows that match a condition	8
Section 1.8: UPDATE All Rows	8
Section 1.9: TRUNCATE TABLE	9
Section 1.10: Retrieve Basic Server Information	9
Section 1.11: Create new table and insert records from old table	9
Section 1.12: Using Transactions to change data safely	10
Section 1.13: Getting Table Row Count	11
Chapter 2: Data Types	12
Section 2.1: Exact Numerics	12
Section 2.2: Approximate Numerics	13
Section 2.3: Date and Time	13
Section 2.4: Character Strings	14
Section 2.5: Unicode Character Strings	14
Section 2.6: Binary Strings	14
Section 2.7: Other Data Types	14
Chapter 3: Converting data types	15
Section 3.1: TRY PARSE	15
Section 3.2: TRY CONVERT	15
Section 3.3: TRY CAST	16
Section 3.4: Cast	16
Section 3.5: Convert	16
Chapter 4: User Defined Table Types	18
Section 4.1: creating a UDT with a single int column that is also a primary key	18
Section 4.2: Creating a UDT with multiple columns	18
Section 4.3: Creating a UDT with a unique constraint:	18
Section 4.4: Creating a UDT with a primary key and a column with a default value:	18
Chapter 5: SELECT statement	19
Section 5.1: Basic SELECT from table	19
Section 5.2: Filter rows using WHERE clause	19
Section 5.3: Sort results using ORDER BY	19
Section 5.4: Group result using GROUP BY	19
Section 5.5: Filter groups using HAVING clause	20
Section 5.6: Returning only first N rows	20
Section 5.7: Pagination using OFFSET FETCH	20
Section 5.8: SELECT without FROM (no data source)	20
Chapter 6: Alias Names in SQL Server	21
Section 6.1: Giving alias after Derived table name	21
Section 6.2: Using AS	21
Section 6.3: Using =	21

Section 6.4: Without using AS	21
Chapter 7: NULLs	22
Section 7.1: COALESCE ()	22
Section 7.2: ANSI NULLS	22
Section 7.3: ISNULL()	23
Section 7.4: Is null / Is not null	23
Section 7.5: NULL comparison	23
Section 7.6: NULL with NOT IN SubQuery	24
Chapter 8: Variables	26
Section 8.1: Declare a Table Variable	26
Section 8.2: Updating variables using SELECT	26
Section 8.3: Declare multiple variables at once, with initial values	27
Section 8.4: Updating a variable using SET	27
Section 8.5: Updating variables by selecting from a table	28
Section 8.6: Compound assignment operators	28
Chapter 9: Dates	29
Section 9.1: Date & Time Formatting using CONVERT	29
Section 9.2: Date & Time Formatting using FORMAT	30
Section 9.3: DATEADD for adding and subtracting time periods	31
Section 9.4: Create function to calculate a person's age on a specific date	32
Section 9.5: Get the current DateTime	32
Section 9.6: Getting the last day of a month	33
Section 9.7: CROSS PLATFORM DATE OBJECT	33
Section 9.8: Return just Date from a DateTime	33
Section 9.9: DATEDIFF for calculating time period differences	34
Section 9.10: DATEPART & DATENAME	34
Section 9.11: Date parts reference	35
Section 9.12: Date Format Extended	35
Chapter 10: Generating a range of dates	39
Section 10.1: Generating Date Range With Recursive CTE	39
Section 10.2: Generating a Date Range With a Tally Table	39
Chapter 11: Database Snapshots	40
Section 11.1: Create a database snapshot	40
Section 11.2: Restore a database snapshot	40
Section 11.3: DELETE Snapshot	40
Chapter 12: COALESCE	41
Section 12.1: Using COALESCE to Build Comma-Delimited String	41
Section 12.2: Getting the first not null from a list of column values	41
Section 12.3: Coalesce basic Example	41
Chapter 13: IF...ELSE	43
Section 13.1: Single IF statement	43
Section 13.2: Multiple IF Statements	43
Section 13.3: Single IF...ELSE statement	43
Section 13.4: Multiple IF... ELSE with final ELSE Statements	44
Section 13.5: Multiple IF...ELSE Statements	44
Chapter 14: CASE Statement	45
Section 14.1: Simple CASE statement	45
Section 14.2: Searched CASE statement	45
Chapter 15: INSERT INTO	46

Section 15.1: INSERT multiple rows of data	46
Section 15.2: Use OUTPUT to get the new Id	46
Section 15.3: INSERT from SELECT Query Results	47
Section 15.4: INSERT a single row of data	47
Section 15.5: INSERT on specific columns	47
Section 15.6: INSERT Hello World INTO table	47
Chapter 16: MERGE	48
Section 16.1: MERGE to Insert / Update / Delete	48
Section 16.2: Merge Using CTE Source	49
Section 16.3: Merge Example - Synchronize Source And Target Table	49
Section 16.4: MERGE using Derived Source Table	50
Section 16.5: Merge using EXCEPT	50
Chapter 17: CREATE VIEW	52
Section 17.1: CREATE Indexed VIEW	52
Section 17.2: CREATE VIEW	52
Section 17.3: CREATE VIEW With Encryption	53
Section 17.4: CREATE VIEW With INNER JOIN	53
Section 17.5: Grouped VIEWS	53
Section 17.6: UNION-ed VIEWS	54
Chapter 18: Views	55
Section 18.1: Create a view with schema binding	55
Section 18.2: Create a view	55
Section 18.3: Create or replace view	55
Chapter 19: UNION	56
Section 19.1: Union and union all	56
Chapter 20: TRY/CATCH	59
Section 20.1: Transaction in a TRY/CATCH	59
Section 20.2: Raising errors in try-catch block	59
Section 20.3: Raising info messages in try catch block	60
Section 20.4: Re-throwing exception generated by RAISERROR	60
Section 20.5: Throwing exception in TRY/CATCH blocks	60
Chapter 21: WHILE loop	62
Section 21.1: Using While loop	62
Section 21.2: While loop with min aggregate function usage	62
Chapter 22: OVER Clause	63
Section 22.1: Cumulative Sum	63
Section 22.2: Using Aggregation functions with OVER	63
Section 22.3: Dividing Data into equally-partitioned buckets using NTILE	64
Section 22.4: Using Aggregation funtions to find the most recent records	64
Chapter 23: GROUP BY	66
Section 23.1: Simple Grouping	66
Section 23.2: GROUP BY multiple columns	66
Section 23.3: GROUP BY with ROLLUP and CUBE	67
Section 23.4: Group by with multiple tables, multiple columns	68
Section 23.5: HAVING	69
Chapter 24: ORDER BY	71
Section 24.1: Simple ORDER BY clause	71
Section 24.2: ORDER BY multiple fields	71
Section 24.3: Custom Ordering	71

Section 24.4: ORDER BY with complex logic	72
Chapter 25: The STUFF Function	73
Section 25.1: Using FOR XML to Concatenate Values from Multiple Rows	73
Section 25.2: Basic Character Replacement with STUFF()	73
Section 25.3: Basic Example of STUFF() function	74
Section 25.4: stuff for comma separated in sql server	74
Section 25.5: Obtain column names separated with comma (not a list)	74
Chapter 26: JSON in SQL Server	76
Section 26.1: Index on JSON properties by using computed columns	76
Section 26.2: Join parent and child JSON entities using CROSS APPLY OPENJSON	77
Section 26.3: Format Query Results as JSON with FOR JSON	78
Section 26.4: Parse JSON text	78
Section 26.5: Format one table row as a single JSON object using FOR JSON	78
Section 26.6: Parse JSON text using OPENJSON function	79
Chapter 27: OPENJSON	80
Section 27.1: Transform JSON array into set of rows	80
Section 27.2: Get key:value pairs from JSON text	80
Section 27.3: Transform nested JSON fields into set of rows	80
Section 27.4: Extracting inner JSON sub-objects	81
Section 27.5: Working with nested JSON sub-arrays	81
Chapter 28: FOR JSON	83
Section 28.1: FOR JSON PATH	83
Section 28.2: FOR JSON PATH with column aliases	83
Section 28.3: FOR JSON clause without array wrapper (single object in output)	83
Section 28.4: INCLUDE NULL VALUES	84
Section 28.5: Wrapping results with ROOT object	84
Section 28.6: FOR JSON AUTO	84
Section 28.7: Creating custom nested JSON structure	85
Chapter 29: Queries with JSON data	86
Section 29.1: Using values from JSON in query	86
Section 29.2: Using JSON values in reports	86
Section 29.3: Filter-out bad JSON text from query results	86
Section 29.4: Update value in JSON column	86
Section 29.5: Append new value into JSON array	87
Section 29.6: JOIN table with inner JSON collection	87
Section 29.7: Finding rows that contain value in the JSON array	87
Chapter 30: Storing JSON in SQL tables	88
Section 30.1: JSON stored as text column	88
Section 30.2: Ensure that JSON is properly formatted using ISJSON	88
Section 30.3: Expose values from JSON text as computed columns	88
Section 30.4: Adding index on JSON path	88
Section 30.5: JSON stored in in-memory tables	89
Chapter 31: Modify JSON text	90
Section 31.1: Modify value in JSON text on the specified path	90
Section 31.2: Append a scalar value into a JSON array	90
Section 31.3: Insert new JSON Object in JSON text	90
Section 31.4: Insert new JSON array generated with FOR JSON query	91
Section 31.5: Insert single JSON object generated with FOR JSON clause	91
Chapter 32: FOR XML PATH	93

[Click here to download full PDF material](#)