**Free text search** A full-text index includes one or more character-based columns in a table. These columns can have any of the following data types: **char**, **varchar**, **nchar**, **nvarchar**, **text**, **ntext**, **image**, **xml**, or **varbinary(max)** and **FILESTREAM**. Each full-text index indexes one or more columns from the table, and each column can use a specific language.

Step 1 - To use the free text search first we need to enable the feature during installation of the SQL Server.

Step 2 – Need to add index on the colom where we want to enable the full text search. Feature will not be enable till we do the step 1



Step 3 – Select an index and click next



Step 4 – Select the colom from the list and click next. Then the selected colom will be able to support full text search



Step 5 – Sample Queries for the full text search.

 select \* from Production.ProductDescription

SELECT Description

FROM Production.ProductDescription

WHERE ProductDescriptionID <> 5 AND

 CONTAINS(Description, 'Top AND spindle')

Sample

Full-text queries use a small set of Transact-SQL predicates (CONTAINS and FREETEXT) and functions (CONTAINSTABLE and FREETEXTTABLE). However, the search goals of a given business scenario influence the structure of the full-text queries. For example:

String should be in double quotes if space in string if we have to use contains.

SELECT Description

FROM Production.ProductDescription

WHERE ProductDescriptionID <> 5 AND

 CONTAINS(Description,”Snap Happy” )

We can also use and /or / <> operation with contains keyword like that CONTAINS(Description,”Snap OR Happy” )

Search with prefix

USE AdventureWorks2012

GO

SELECT Description, ProductDescriptionID

FROM Production.ProductDescription

WHERE CONTAINS (Description, '"top\*"' )

GO

The following example searches for any form of "foot" ("foot," "feet," and so on) in the Comments column of the ProductReview table in the AdventureWorks database:

USE AdventureWorks2012

GO

SELECT Comments, ReviewerName

FROM Production.ProductReview

WHERE CONTAINS (Comments, 'FORMSOF(INFLECTIONAL, "foot")')

GO

Configure and Manage Thesaurus Files for Full-Text Search – This needs to explore yet.

SQL Server Full-Text Search queries can search for synonyms of user-specified terms through the use of a Full-Text Search thesaurus. Each thesaurus defines a set of synonyms for a specific language. By developing a thesaurus tailored to your full-text data, you can effectively broaden the scope of full-text queries on that data.

File Location of Synonyms C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\FTData

File name – tsenu.xml and tsglobal.xml

**Sp\_lock –** sp\_lock maintains lock to prevent concurrent use of resources by different transactions.

Notes- no practical example done yet.

Syntax - sp\_lock [ [ @spid1 = ] 'session ID1' ] [ , [@spid2 = ] 'session ID2' ]

[ ; ]

## Examples

### A. Listing all locks

USE master;

GO

EXEC sp\_lock;

GO

### B. Listing a lock from a single-server process

### The following example displays information, including locks, about process ID 53.

USE master;

GO

EXEC sp\_lock 53;

GO

**Sp\_who** Provides information about current users, sessions, and processes in an instance of the Microsoft SQL Server Database Engine. The information can be filtered to return only those processes that are not idle, that belong to a specific user, or that belong to a specific session.

sp\_who [ [ @loginame = ] 'login' | session ID | 'ACTIVE' ]

USE master;

GO

EXEC sp\_who 'janetl';

GO

**NOLOCK** –

* The NOLOCK hint allows SQL to read data from tables by ignoring any locks and therefore not get blocked by other processes.
* This can improve query performance by removing the blocks, but introduces the possibility of dirty reads.

## **Example of SQL Server NOLOCK**

Here is a query that returns all of the data from the Person.Contact table. If I run this query I can see there is only one record that has a Suffix value for ContactID = 12.

SELECT \* FROM Person.Contact WHERE ContactID < 20



Let's say another user runs the below query in a transaction. The query completes and updates the records, but it is not yet committed to the database so the records are locked.

-- run in query window 1

BEGIN TRAN

UPDATE Person.Contact SET Suffix = 'B' WHERE ContactID < 20

-- ROLLBACK or COMMIT

If you run the same query that was run above again, you will notice that it never completes, because the UPDATE statement run in "query window 1" has not yet been committed, so the locks are not released.

-- run in query window 2

SELECT \* FROM Person.Contact WHERE ContactID < 20

If I run **sp\_who2**I can see that the SELECT statement is being blocked. I will need to either cancel this query or COMMIT or ROLLBACK the query in window 1 for this to complete. For this example I am going to cancel the SELECT query.

To get around the locked records, I can use the NOLOCK hint as shown below and the query will complete even though the query in window 1 is still running and has not been committed or rolled back.

-- run in query window 2

SELECT \* FROM Person.Contact WITH (NOLOCK) WHERE ContactID < 20

**Features**

1. SQL server agent service in MSSQL

2. mail configuration to send success/Failure of any SQL server agent event

3. SQL profiler own monitoring tool

4. Can create Linked server between two or more servers

5. MSSQL is licensed and they provide support on enterprise edition.

6. Mirroring and always High availability feature

7. SSAS and SSRS tool also Microsoft provide with MSSQL for analysis tool and reporting tool.

Identify Item - How to create schema, No LOCK , Commit rollback and @@trancount

**Best Practices**

How to create schema – example CREATE TABLE [new].[DemoSchema] (ID INT IDENTITY(1, 1), Name VARCHAR(20) );

* Always qualify objects by owner. (Get more details) **Example** – whenever stored procedure run then a  cached plan is generated for the fast execution next time. always run by dbo.mystoredproc – Lets suppose if another user harry run the same procedure with only name mystoredproc then it find the object first then cached the another plan which will cause another cache If an existing plan is found, SQL Server reuses the cached plan and does not actually compile the stored procedure. However, the lack of owner-qualification forces SQL to perform a second cache lookup
* Do not use GOTO. –(Get more details in example)

Use of the GOTO statement is generally considered to be poor programming practice and is not recommended. Extensive use of GOTO tends to lead to unreadable code especially when procedures grow long

1DECLARE @Counter int;

 2SET @Counter = 1;

 3WHILE @Counter < 10

 4BEGIN

 5 SELECT @Counter

 6 SET @Counter = @Counter + 1

 7 IF @Counter = 4 GOTO Branch\_One -- Jumps to the first branch.

 8 IF @Counter = 5 GOTO Branch\_Two -- This will never execute.

 9END

10Branch\_One:

11 SELECT 'Jumping To Branch One.'

12 GOTO Branch\_Three; --This will prevent Branch\_Two from executing.

13Branch\_Two:

14 SELECT 'Jumping To Branch Two.'

15Branch\_Three:

16 SELECT 'Jumping To Branch Three.

* Avoid CURSOR use because it's significantly slower. If necessary, always declare the correct type of cursor (FAST\_FORWARD).
* Avoid SELECT INTO for populating temp tables. Create the table then use INSERT SELECT. –

SELECT \* INTO #TempOrdersTable

FROM Orders ( Not recommended)

Create and insert

INSERT INTO #TempPersonTable (Recommended)

VALUES

( 'Watson', 'Juan', 'Cleveland'),

( 'Baker', 'Dwayne', 'Fort Wayne'),

( 'Walker', 'Eric', 'Tucson'),

( 'Peterson', 'Bob', 'Indianapolis');

SELECT \*

FROM #TempPersonTable;

* Always use ANSI join syntax.

Non ANSI Syntax

SELECT a.name,

 a.empno,

 b.loc

 FROM tab a,

 tab b

 WHERE a.deptno = b.deptno(+)

 AND a.empno = 190;

ANSI Syntax (recommended)

SELECT a.name,

 a.empno,

 b.loc

 FROM tab a,

LEFT OUTER JOIN tab b

 ON a.deptno = b.deptno

 WHERE a.empno = 190;

* Always check for object existence.

IF EXISTS (SELECT \* FROM dbo.sysobjects WHERE id = object\_id (N'[dbo].[IsSomething]') AND OBJECTPROPERTY(id, N'IsFunction') = 1)

DROP function IsSomething

GO

* Use SCOPE\_IDENTITY() instead of @@IDENTITY.

SCOPE\_IDENTITY and @@IDENTITY return the last identity values that are generated in any table in the current session. However, SCOPE\_IDENTITY returns values inserted only within the current scope; @@IDENTITY is not limited to a specific scope.

* Always check @@TRANCOUNT and commit/rollback as necessary.

when u don't use @@trancount, the error message of nested transaction stored procedure does not return the exact cause of error just reurtn "The rollback transaction request has no corresponding begin transaction",otherwise it gives exact cause of error, so its easy to handle the error with proper syntax.

`

Example begin tran

begin try

 ... just several lines of sql ...

 if @@trancount > 0 commit tran

end try

begin catch

 if @@trancount > 0 rollback tran

end catch

* Order [DML](https://en.wikipedia.org/wiki/Data_manipulation_language%22%20%5Co%20%22w%3AData%20manipulation%20language) to avoid deadlocks. ( Need more time to explain)
* Always check @@ERROR and @@ROWCOUNT by assigning to a variable.

Returns the error number for the last Transact-SQL statement executed. Returns 0 if the previous Transact-SQL statement encountered no errors.

GO

UPDATE HumanResources.EmployeePayHistory

 SET PayFrequency = 4

 WHERE BusinessEntityID = 1;

IF @@ERROR <> 0

 BEGIN

 PRINT N'A check constraint violation occurred.';

 END

GO

* Always check sp return values. – While executing the sp check the return values with the type they returned if any
* Do not create cross-database dependencies. – Avoid cross DB dependency like inner joins or so. It is slow than using with in the same DB – (Need more time to show with example)
* Avoid table value UDF – performance problems.

A table function, also called a table-valued function (TVF), is a user-defined function that returns a table. You can use a table function anywhere that you can use a table. Table functions behave similarly to views, but a table function can take parameters.

CREATE OR REPLACE TABLE FUNCTION mydataset.names\_by\_year(y INT64)
AS
  SELECT year, name, SUM(number) AS total
  FROM `bigquery-public-data.usa\_names.usa\_1910\_current`
  WHERE year = y
  GROUP BY year, name

* Avoid dynamic SQL – if necessary use sp\_executesql over EXEC.

Static or Embedded SQL are SQL statements in an application that do not change at runtime and, therefore, can be hard-coded into the application. **Dynamic SQL is SQL statements that are constructed at runtime**; for example, the application may allow users to enter their own queries.

* Avoid using NULL values.
* In some cases they are not indexed.
* They make join syntax more difficult.
* They need special treatment for comparisons.

For string columns it might be appropriate to use "N/A", or "N/K" as a special value that helps distinguish between different classes of what could otherwise be NULL, but that's tricky to do for numerics or dates -- special values are generally tricky to use, and it may be better to add an extra column (eg. for date\_of\_birth you might have a column that specifies "reason\_for\_no\_date\_of\_birth", which can help the application be more useful.

For many cases where data values are genuinely unknown or not relevant they can be entirely appropriate of course -- date\_of\_death is a good example, or date\_of\_account\_termination.

Sometimes even these examples can be rendered irrelevant by normalising events out to a different table, so you have a table for "ACCOUNT\_DATES" with DATE\_TYPES of "Open", "Close", etc.

* Always specify columns; try to avoid "SELECT \*". Exceptions include these two cases: "WHERE EXISTS (SELECT \* ...)" and aggregate functions.

If we dont need all column then we should use column name to get all coloms.

**Stored Procedures**

**Create procedure**

CREATE PROCEDURE dbo.uspGetAddress @City nvarchar(30)

AS

SELECT \*

FROM Person.Address

WHERE City = @City

GO

EXEC uspGetAddress @City = 'New York'

List procedure

SELECT name AS [Name],

       SCHEMA\_NAME(schema\_id) AS schema\_name,

       type\_desc,

       create\_date,

       modify\_date

FROM sys.objects

WHERE type ='p'

### **Use sys.objects system catalog view**

SELECT name AS [Name],

 SCHEMA\_NAME(schema\_id) AS schema\_name,

 type\_desc,

 create\_date,

 modify\_date

FROM sys.objects

WHERE type ='u'

COALESCE function use to remove null

## **SQL SCOPE\_IDENTITY() function**

SELECT IDENT\_CURRENT('users') AS IdentityValue

SELECT TOP (10) IDENTITY( INT, 100, 2) AS NEW\_ID,

 [PersonType],

 [NameStyle],

 [Title],

 [FirstName],

 [MiddleName],

 [LastName],

 [Suffix]

INTO TEMPTABLE

FROM [AdventureWorks2019].[Person].[Person];

sp\_help 'TEMPTABLE'

LAG provides access to a row at a given physical offset that comes before the current row. Use this analytic function in a SELECT statement to compare values in the current row with values in a previous row.

DENSE\_RANK

LAG

COALESCE

sp\_who2

sp\_lock

NOLOCK

NTILE

FIRST\_VALUE

LAST\_VALUE, CONCAT\_WS, @@ROWCOUNT