# **Subquery to CTE**

# **2. T-SQL Errors and Reactions**

**Let's say we're executing a stored procedure called users stored procedure *A* that sits on the server.**

* The job of this stored procedure is to call to other stored procedures ***B*** and ***C.***
* In Store procedure B has three statements inside of it once it's done it's going to call stored procedure C.

**If we encounter an error in statement number two of stored procedure *B* should it go onto statement three or should the entire stored procedure *B* fail and if it *B* fails should store procedure A once it gets a failure back from *B* , continue to call C or should it say we're going to cancel the whole thing?**

**It depends on the type of error.**

1. If statement two does fail there are situations where it will go on to statement three and succeed and stored procedure ***B*** will return a success that reaction is known as a ***statement termination***.
2. In another situation it might cause the entire stored procedure ***B*** to fail but user stored procedure ***A*** will still continue on and call ***C*** to do its work that's known as ***scope abortion***.
3. Another situation would be known as ***batch abortion*** in which the entire call from the client no longer works that would cause stored procedure a to completely fail and not continue on at all.

**If you encounter any error of with severity level 11 or higher, we can fire batch termination by using transact abort.**

SET *XACT\_ABORT* ON;

# **07. Difference Temp Table and Table Variable – Effect of Transaction**

***MAXDOP* controls intra-query parallelism in the database engine.**

* Higher MAXDOP values generally result in more parallel threads per query, and faster query execution.

**In general, if the database engine chooses to execute a query using parallelism, execution time is faster**.

* However, excess parallelism can consume additional processor resources without improving query performance.
* At scale, excess parallelism can negatively affect query performance for all queries executing on the same database engine instance.
* Traditionally, setting an upper bound for parallelism has been a common performance tuning exercise in SQL Server workloads.

|  |  |
| --- | --- |
| MAXDOP | Behavior |
| = 1 | **The database engine uses a single serial thread to execute queries. Parallel threads are not used.** |
| > 1 | **The database engine sets the number of additional schedulers to be used by parallel threads to the MAXDOP value, or the total number of logical processors, whichever is smaller.** |
| = 0 | **The database engine sets the number of additional schedulers to be used by parallel threads to the total number of logical processors or 64, whichever is smaller.** |

# **12. Introduction to Hierarchical Query**

**A CTE can be thought of as a temporary result set and are similar to a derived table in that it is not stored as an object and lasts only for the duration of the query.**

**A CTE is generally considered to be more readable than a derived table and does not require the extra effort of declaring a Temp Table while providing the same benefits to the user.**

**However; a CTE is more powerful than a derived table as it can also be self-referencing, or even referenced multiple times in the same query.**

**Building a Recursive CTE**

**In the following examples, you will learn how to harness the power of a recursive CTE query by fulfilling a common business requirement, retrieving hierarchical data. By the time the final query is complete you will be able to easily determine how many levels from the top executive each employee is.**

## **A recursive CTE requires four elements in order to work properly.**

1. **Anchor query (runs once and the results ‘seed’ the Recursive query)**
2. **Recursive query (runs multiple times and is the criteria for the remaining results)**
3. **UNION ALL statement to bind the Anchor and Recursive queries together.**
4. **INNER JOIN statement to bind the Recursive query to the results of the CTE.**

WITH MyCTE

AS ( SELECT EmpID, FirstName, LastName, ManagerID

FROM Employee

WHERE ManagerID IS NULL

UNION ALL

SELECT EmpID, FirstName, LastName, ManagerID

FROM Employee

INNER JOIN MyCTE ON Employee.ManagerID = MyCTE.EmpID

WHERE Employee.ManagerID IS NOT NULL)

SELECT \* FROM MyCTE

# **13. Introduction to Basics of a Query Hint**

**Query hints specify that the indicated hints should be used throughout the query.**

**Query hints affect all operators in the statement and are implemented using the OPTION clause.**

DECLARE @Type VARCHAR ( 50 )

SET @Type = 'Business'

SELECT \*

FROM Customer

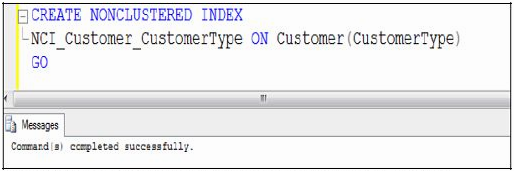
WHERE CustomerType = @Type

OPTION (OPTIMIZE FOR ( @Type = 'Business' ));

**Cautionary Note:**

**Because the SQL Server Query Optimizer typically selects the best execution plan for a query, it is highly recommended that hints be used as a last resort for experienced developers and database administrators to achieve the desired results.**

## **Create an Index for improved Optimization**

In the following examples, the Customer Table has had a non-clustered index placed on the ***CustomerType*** field, as shown in the screenshot below:

**This index should help query performance if the query is selective enough.**

* That is, if the Query Optimizer gets the right information in time to make this kind of a decision it will perform a Seek which is much faster than a Scan.
* Variables often confuse the Query Optimizer in certain ways

**The Query Optimizer doesn’t know what value the @Type variable holds until runtime, and does not check statistics for variables before a query runs.**

* When in doubt, the Query Optimizer will always choose a scan.

### **Implementing the Query Hint**

**When predicating on values from variables you may need to give the query a hint, letting the Query Optimizer know the best way to execute the query.**

**If your @Type variable is most often set to the value of ‘Business’ it would be smarter to optimize this query to perform a Seek based on this value.**

# **15. Introduction to XML Data Type Methods**

**The XML data type was first introduced with SQL Server 2005.**

* This data type continues with SQL Server 2008 where expanded XML features are available, most notably is the power of the XQuery language to analyze and query the values contained in your XML instance.

**There are five XML data type methods available in SQL Server 2008:**

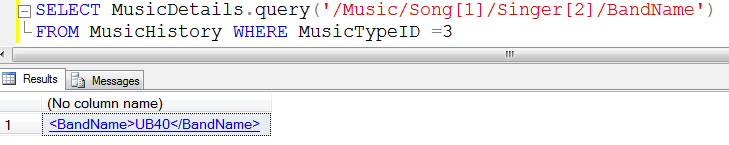
1. **query**() – Used to extract XML fragments from an XML data type.
2. **value**() – Used to extract a single value from an XML document.
3. **exist**() – Used to determine if a specified node exists. Returns 1 if yes and 0 if no.
4. **modify**() – Updates XML data in an XML data type.
5. **node**() – Shreds XML data into multiple rows (not covered in this blog post).

**XML data type methods require an XPath expression as one of the (or the only) XQuery parameter(s), but not all of the methods return data.**

* Some of these methods simply analyze the data at that level and return a status to you. In order to be proficient with any of the XML data type methods, it is important for you to become familiar with using XPath expressions.

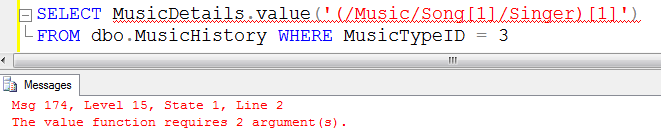
## **The query() Method**

**This method basically needs an XPath expression in the XQuery parameter and returns an XML data type.**

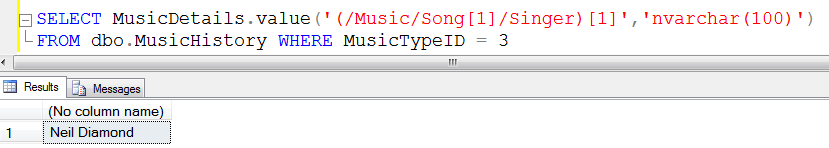
* The XPath expression (‘/Music/Song[1]/Singer[2]/BandName’) specifies that we want to navigate to the BandName element of the second Singer of the first Song.
* The query( ) method returns the XML fragment containing everything between (and including) the beginning and ending tags of that BandName element, which in this example is UB40.

## **The value() Method**

**It returns just your data (without the metadata – no element tags) and gives you the freedom to specify any data type you would like. In other words, if you are pulling from a element, then you might want the result returned as a Money or a Decimal data type.**

 **If you are pulling from the element, you might specify that the returning data should be a varchar data type.**

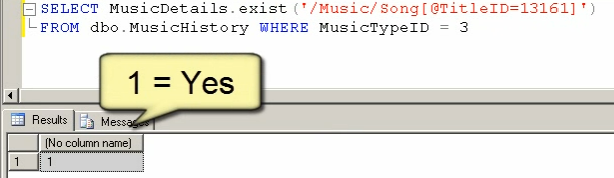
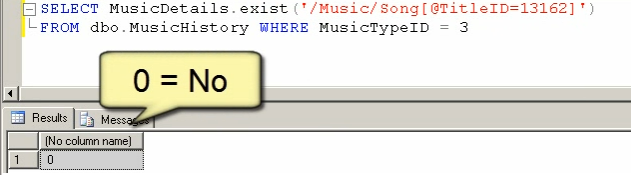
Observe that XQuery gives us the freedom to specify data types which are compatible with character data (e.g., char(20), varchar(max)).



## **The exist() Method**

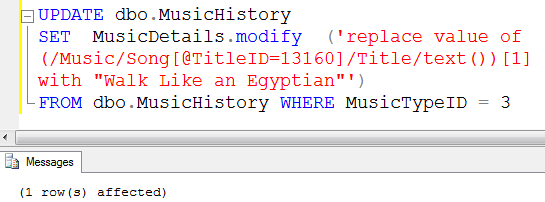
**It will check for the existence and even the value of the XPath expression you specify.**

**In this example you will use the exist( ) method to determine if a particular song is in the catalogue.**

* The song TitleID you are looking for is 13161. As you can see from the screenshot, this TitleID does exist as indicated by the return value of 1 (numeric value for ‘Yes’).

## **The modify() Method**

**The modify( ) method allows you to change values directly in your XML stream.**

* Like all other XML data type methods, it needs an XPath parameter to know which value to change.
* However, unlike the other methods, ***modify( )*** works with an ***UPDATE*** statement (it will not work with a SELECT statement).
* Also, modify( ) can only work with one data value at a time, which is a mathematical and programming concept known as a singleton.

# **16. Standard Reports from SQL Server Management Studio**

**SQL Server management Studio 2012 is wonderful tool and has many different features.**

* Many times, an average user does not use them as they are not aware about these features. Today, we will learn one such feature.
* SSMS comes with many inbuilt performance and activity reports, but we do not use it to the full potential.

***Connect to SQL Server Node >> Right Click on it >> Go to Reports >> Click on Standard Reports >> Pick Any Report.***

# **17. Tricks to Replace SELECT \* with Column Names**

**SELECT \* has many disadvantages.**

* Retrieves unnecessary columns and increases network traffic
* When a new columns are added views needs to be refreshed manually
* Leads to usage of sub-optimal execution plan
* Uses clustered index in most of the cases instead of using optimal index
* It is difficult to debug.

**There are two quick tricks I have discussed in the video which explains how users can avoid using SELECT \* but instead list the column names.**

1. **Drag the columns folder from SQL Server Management Studio to Query Editor**
2. **Right Click on Table Name >> Script Table AS >> SELECT To… >> Select option**

# **18. Importing CSV into SQL Server**

**Importing data into database is one of the most important tasks.**

## **Bulk Insert**

BULK  
INSERT CSVTest  
FROM 'c:\csvtest.txt'  
WITH  
(  
FIELDTERMINATOR = ',',  
ROWTERMINATOR = '\n'  
)  
GO  
--Check the content of the table.  
SELECT \*  
FROM CSVTest  
GO  
--Drop the table to clean up database.  
DROP TABLE CSVTest  
GO

# 26. Effect of Collation on Resultset

## **What is collation?**

**Collation is nothing but a set of rules that are predefined in SQL Server that determine how the data in SQL Server are stored, retrieved and compared.**

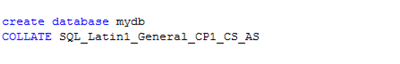
**Mainly there are various rules or collations that exist in SQL Server but we need to know the following 2 main collations.**

1. SQL\_Latin1\_General\_CP1\_CI\_AS
2. SQL\_Latin1\_General\_CP1\_CS\_AS

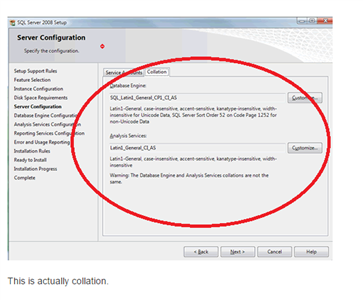
**CS is case sensitive,**

* So by default in SQL Server the collation is case insensitive and that's why it does not worry about any cases, so it is case insensitive.
* Since SQL Server is case insensitive all the databases in it are also case insensitive.

**Here we have a requirement in my project for a log in the form and I need to match the data. If only the data is in the the same cases then what I will do is fetch the data and check if only the spelling is correct. SQL Server will not check any cases.**

**Thus to overcome this problem we need to use the collate manually in our database or in our column of a table.**

**Do you know why it is case insensitive?**

* This is because of the following things that you are choosing during the installation of SQL Server.

# 29. Query to Find Row and Index Count of Database Tables

SELECT [schema\_name] = s.name,

table\_name = o.name,

MAX(i1.type\_desc) ClusteredIndexorHeap,

MAX(COALESCE(i2.NCIC,0)) NoOfNonClusteredIndex,

p.rows

FROM sys.indexes si

RIGHT JOIN sys.objects o ON si.[object\_id] = o.[object\_id]

INNER JOIN sys.schemas s ON o.[schema\_id] = s.[schema\_id]

LEFT JOIN sys.partitions p ON p.OBJECT\_ID = o.OBJECT\_ID AND p.index\_id IN (0,1)

LEFT JOIN sys.indexes i1 ON si.OBJECT\_ID = i1.OBJECT\_ID AND i1.TYPE IN (0,1)

LEFT JOIN (SELECT object\_id,COUNT(Index\_id) NCIC

FROM sys.indexes

WHERE type = 2

GROUP BY object\_id) I2

ON si.OBJECT\_ID = i2.OBJECT\_ID

WHERE o.TYPE IN ('U')

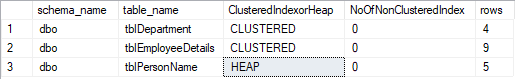
GROUP BY s.name, o.name, p.rows

ORDER BY schema\_name, table\_name

**How many users created tables are there in the database?**

**How many non clustered indexes each of the tables in the database have?**

**Is table Heap or has clustered index on it?**

**How many rows each of the tables is contained in the database?**

## **Heaps**

**A heap is a table that is stored without any underlying order.**

* When rows are inserted into a heap, there is no way to ensure where the pages will be written nor are those pages guaranteed to remain in the same order as the table is written to or when maintenance is performed against it.

**The heap is comprised of an Index Allocation Map (IAM) that points to all pages within the heap.**

* Each page will contain as many rows of data as will fit, as they are written. Within the heap, there is no linking or organization between the pages.
* All reads and writes must consult the IAM first to read all pages within a heap.

**While there are many other considerations about how a heap is stored and how its data is managed, the most important aspect of it is lack of order.**

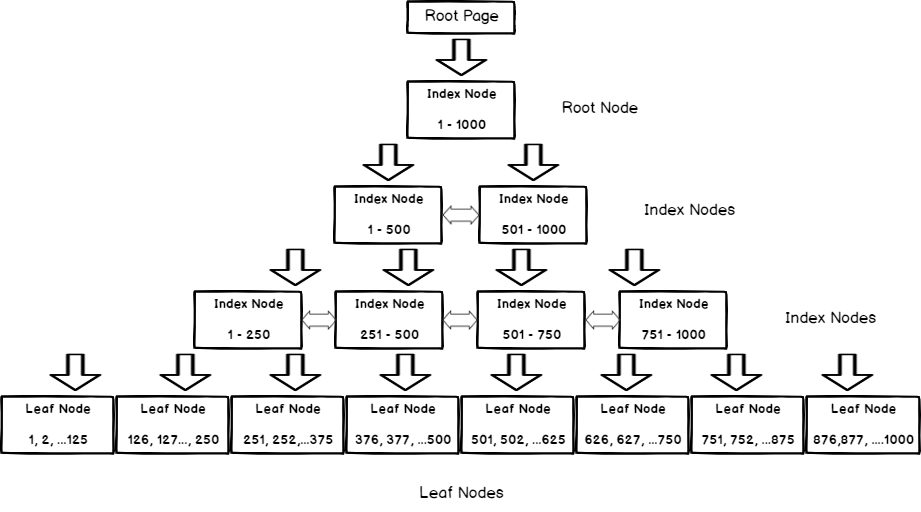
* The primary reason why heaps behave as they do will be that the rows are stored without any specified order.
* This fact will have generally negative implications on read and write operations.

## **Basic heap storage structureCluster Index**

**The alternative to an unordered heap is to define a table with a clustered index.**

* This index provides an innate ordering for the table it is defined on and follows whatever column order the index is defined on.
* In a clustered index, when rows are inserted, updated, or deleted, the underlying order of data is retained.

**A clustered index is stored as a binary tree (B-tree for short).**

* This structure starts with a root node and branches out in pairs to additional nodes until enough exists to cover the entire table’s worth of values for the index.
* In addition to providing an ordering of data, the nodes of the B-tree provide pointers to the next and previous rows in the data set.