Some Sample Code of SQL Best Practices

1. **Sp\_ ExecuteSQL**

Eg for using sp\_executesql :

DECLARE @IntVariable int;

DECLARE @SQLString nvarchar(500);

DECLARE @ParmDefinition nvarchar(500);

SET @SQLString ='select ProductSubName from tblM\_ProSubCategory

where ProductSubID=@value'

SET @ParmDefinition ='@value int'

SET @IntVariable = 32

execute sp\_executesql @SQLString, @ParmDefinition,

@value = @IntVariable;

1. **Transaction**

--- COMMITTING INTERNAL TRANSACTION AND ROLLBACK THE EXTERNAL TRANSACTION--

CREATE Table sampleTb (Firstcol int Primary key ,secondcol char(10) NOT NULL)

GO

BEGIN TRAN Outtran

INSERT INTO sampleTb VALUES(5,'FIFTH')

INSERT INTO sampleTb VALUES(6,'SIXTH')

BEGIN TRAN Intran

INSERT INTO sampleTb VALUES(7,'SEVENTH')

INSERT INTO sampleTb VALUES(8,'EIGHTH')

COMMIT TRAN Intran

ROLLBACK TRAN Outtran

SELECT \* FROM sampleTb

OUTPUT : NO OUTPUT

--COMMITING THE OUTER TRAN AND ROLLBACK THE INNERTRAN---

CREATE Table sampleTb (Firstcol int Primary key ,secondcol char(10) NOT NULL)

GO

BEGIN TRAN Outtran

INSERT INTO sampleTb VALUES(5,'FIFTH')

INSERT INTO sampleTb VALUES(6,'SIXTH')

BEGIN TRAN Intran

INSERT INTO sampleTb VALUES(7,'SEVENTH')

INSERT INTO sampleTb VALUES(8,'EIGHTH')

ROLLBACK TRAN Intran

COMMIT TRAN Outtran

SELECT \* FROM sampleTb

OUTPUT: ALL THE 4 ROWS

----------Orphaned transactions---

CREATE Table sampleTb (Firstcol int Primary key, secondcol char (10) NOT NULL)

GO

BEGIN TRAN Outtran

INSERT INTO sampleTb VALUES (11,'ELEVEN')

INSERT INTO sampleTb VALUES (12, null)

COMMIT TRAN Outtran

SELECT \* FROM sampleTb

--Correct code---

CREATE Table sampleTb (Firstcol int Primary key, secondcol char (10) NOT NULL)

GO

BEGIN TRAN Outtran

INSERT INTO sampleTb VALUES (11,'ELEVEN')

INSERT INTO sampleTb VALUES (12,'TWE')

IF @@ERROR <>0

ROLLBACK Tran Outtran

ELSE

COMMIT TRAN Outtran

SELECT \* FROM sampleTb

The @@TRANCOUNT function records the current transaction nesting level. Each BEGIN TRANSACTION statement increments @@TRANCOUNT by one. Each COMMIT TRANSACTION or COMMIT WORK statement decrements @@TRANCOUNT by one. A ROLLBACK TRANSACTION statement that does not have a transaction name rolls back all nested transactions and decrements @@TRANCOUNT to 0. A ROLLBACK TRANSACTION that uses the transaction name of the outermost transaction in a set of nested transactions rolls back all the nested transactions and decrements @@TRANCOUNT to 0. When you are unsure if you are already in a transaction, SELECT @@TRANCOUNT to determine if it is 1 or more. If @@TRANCOUNT is 0 you are not in a transaction.

1. **Index**

DBCC – database console command

DBCC commands are generally used to check the physical and logical consistency of a database

DBCC showcontig

* Use the page count reported by DBCC SHOWCONTIG to get an idea of the size of the indexes (each page is 8 KB in size).
* High values for logical scan fragmentation can lead to degraded performance of index scans.
* Consider defragmenting indexes with 20 percent or more logical fragmentation.
* Low values for average page density can result in more pages that must be read to satisfy a query. Reorganizing the pages so they have a higher page density can result in less I/O to satisfy the same query.
* DBCC INDEXDEFRAG  
      ( { database\_name | database\_id | 0 }   
          , { table\_name | table\_id | 'view\_name' | view\_id }   
          , { index\_name | index\_id }   
      )    [ WITH NO\_INFOMSGS ]
* database\_name | database\_id | 0
* Is the database for which to defragment an index. Database names must conform to the rules for identifiers. For more information, see [Using Identifiers](http://msdn.microsoft.com/en-us/library/aa223962(SQL.80).aspx). If 0 is specified, then the current database is used.
* table\_name | table\_id | 'view\_name' | view\_id
* Is the table or view for which to defragment an index. Table and view names must conform to the rules for identifiers.
* index\_name | index\_id
* Is the index to defragment. Index names must conform to the rules for identifiers.
* WITH NO\_INFOMSGS
* Suppresses all informational messages (with severity levels from 0 through 10).
* DBCC DBREINDEX
* Rebuilds one or more indexes for a table in the specified database.
* DBCC DBREINDEX
* (
* table\_name
* [ , index\_name [ , fillfactor ] ]
* )
* [ WITH NO\_INFOMSGS ]
* table\_name
* Is the name of the table containing the specified index or indexes to rebuild. Table names must follow the rules for [identifiers](http://msdn.microsoft.com/en-us/library/ms175874.aspx).
* index\_name
* Is the name of the index to rebuild. Index names must comply with the rules for identifiers. If index\_name is specified, table\_name must be specified. If index\_name is not specified or is " ", all indexes for the table are rebuilt.
* fillfactor
* Is the percentage of space on each index page for storing data when the index is created or rebuilt. fillfactor replaces the fill factor when the index was created, becoming the new default for the index and for any other nonclustered indexes rebuilt because a clustered index is rebuilt. When fillfactor is 0, DBCC DBREINDEX uses the fill factor value last specified for the index. This value is stored in the sys.indexes catalog view.
* If fillfactor is specified, table\_name and index\_name must be specified. If fillfactor is not specified, the default fill factor, 100, is used. For more information, see [Fill Factor](http://msdn.microsoft.com/en-us/library/ms177459.aspx).
* WITH NO\_INFOMSGS
* Suppresses all informational messages that have severity levels from 0 through 10.
* http://i.msdn.microsoft.com/Global/Images/clear.gif
* Unless NO\_INFOMSGS is specified (the table name must be specified), DBCC DBREINDEX always returns:
* DBCC execution completed. If DBCC printed error messages, contact your system administrator.
* EXAMPLE:
* --CREATION OF TABLE---
* create table employee(
* ID int NOT NULL,
* name nvarchar (10),
* salary int,
* doj datetime,
* city nvarchar (10),
* region char (1))
* GO
* --- INSERTION OF ROWS---
* insert into employee (ID, name, salary, doj, city, region)
* values (1, 'Jason', 40420, '02/01/94', 'New York', 'W')
* GO
* insert into employee (ID, name, salary, doj, city, region)
* values (2, 'Robert',14420, '01/02/95', 'Vancouver','N')
* GO
* insert into employee (ID, name, salary, doj, city, region)
* values (3, 'Celia', 24020, '12/03/96', 'Toronto', 'W')
* GO
* insert into employee (ID, name, salary, doj, city, region)
* values (4, 'Linda', 40620, '11/04/97', 'New York', 'N')
* GO
* insert into employee (ID, name, salary, doj, city, region)
* values (5, 'David', 80026, '10/05/98', 'Vancouver','W')
* GO
* --- INDEX CREATION---
* select \* from employee
* DBCC SHOWCONTIG
* ALTER TABLE employee ADD PRIMARY KEY(ID)
* SP\_HELPINDEX employee
* CREATE INDEX IDX\_Name ON employee(name)
* SP\_HELPINDEX employee
* DBCC SHOWCONTIG
* --to display the actual page information for the table –
* DBCC TRACEON(3604)
* Declare @DBID Int, @TableID Int
* Select @DBID = db\_id(), @TableID = object\_id('employee')
* DBCC ind(@DBID, @TableID, -1)
* GO
* ---INDEX DEFRAGMENTATION---
* --REBUILDING ONE INDEX--
* DBCC DBREINDEX (employee,IDX\_Name) WITH NO\_INFOMSGS
* --REBUILDING ALL INDEXES--
* DBCC DBREINDEX (employee,'')
* --USING INDEXDEFRAG---
* DBCC INDEXDEFRAG(INDUS,employee,IDX\_Name)
* PagePID is the physical page numbers used to store the table. In this case, three pages are currently used to store the data.
* IndexID is the type of index,
* Where:
* 0 – Datapage
* 1 – Clustered Index
* 2 – Greater and equal to 2 is an Index page (Non-Clustered Index and ordinary index),
* PageType tells you what kind of data is stored in each database,
* Where:
* 10 – IAM (Index Allocation MAP)
* 1 – Datapage
* 2 – Index page