

```

/*****
BEGIN TABLE SCRIPT CREATION (this header is repeated at the end)
*****/
declare @ScriptTable TABLE
(Script varchar(7500), Table_Name varchar(255), ScriptRow int )

-----
-- ** Add the create table statement and the columns for the tables
-----

set nocount ON

insert into @ScriptTable
select Script, Table_Name, ScriptRow from
(
select space(30) +
case when [Column_ID] = 1 then ' ' else ', ' end +
' [' + Column_Name +'] [' + Column_Type +'] ' +
case
when Column_Type in ('decimal', 'numeric') then '(' + [Column_Precision] + ', ' + [Column_Scale] + ')'
when Column_Type in ('nvarchar', 'nchar', 'varchar', 'char', 'varbinary', 'binary') then '(' + [Column_Length] + ')'
else ''
end
+ ' ' + Identity_Script + ' ' + Column_Nullable

as [Script], [Table_Name], 1000+[Column_ID] as [ScriptRow]
--, *
from(
select
SO.id as [Table_ID],
SO.name as [Table_Name],
AC.[column_id] as [Column_ID],
AC.[Name] as [Column_Name],
type_name(AC.system_type_id) as [Column_Type],
coalesce(SC.text, '') as [Column_Formula],

case
when AC.[max_length] < 0 then 'max'
when type_name(AC.system_type_id) in ('nchar', 'nvarchar') then cast(AC.[max_length]/2 as varchar(10))
else cast(AC.[max_length] as varchar(10)) end as [Column_Length],
--convert(int, AC.[max_length]) as [Column_Length],

case when type_name(AC.system_type_id) in ('tinyint', 'smallint', 'decimal', 'int', 'real', 'money', 'float', 'numeric', 'smallmoney') then
cast(AC.precision as varchar(20)) else '' end as [Column_Precision],

case when type_name(AC.system_type_id) in ('tinyint', 'smallint', 'decimal', 'int', 'real', 'money', 'float', 'numeric', 'smallmoney') then
cast(AC.scale as varchar(20)) else '' end as [Column_Scale],

CASE WHEN AC.[is_nullable] = 1 THEN 'NULL' ELSE 'NOT NULL' END as [Column_Nullable],
AC.[is_identity] as [Is_Identity],
CASE WHEN AC.[is_replicated] = 0 AND ac.[is_identity] = 1 THEN 'NOT FOR REPLICATION' ELSE '' END as [Column_Replication],
AC.[is_ansi_padded] as [Is_ANSI_Padded],
coalesce(IC.[name], '') as [Identity_Name],
CASE WHEN IC.[name] is not null THEN cast(Ident_Seed(SO.name) as varchar(10)) ELSE '' END as [Identity_Seed],
CASE WHEN IC.[name] is not null THEN cast(Ident_Incr(SO.name) as varchar(10)) ELSE '' END as [Identity_Increment],

CASE WHEN IC.[name] is not null THEN 'IDENTITY(' + cast(Ident_Seed(SO.name) as varchar(10)) + ', ' + cast(Ident_Incr(SO.name) as
varchar(10)) + ')' ELSE '' END as [Identity_Script]

from [sys].[all_columns] AC
left join [syscomments] SC on AC.[object_id] = SC.[id] and AC.[column_id] = SC.[number]
left join [sysobjects] SO on AC.[object_id] = SO.id

```

```

left join [sys].[information_schema].[columns] IC on AC.[object_id] = IC.[object_id] and AC.[column_id] = IC.[column_id]
where SO.xtype = 'U'
) info

union -- Add in the opening part
select 'CREATE TABLE [DBO].[' + SO.[Name] + '] (' as [Script], SO.[Name] as [Table_Name], 1000 as [ScriptRow] from [sysobjects] SO
where xtype = 'U'

union -- Add in the closing part
select ')' as [Script], SO.[Name] as [Table_Name], 2001 as [ScriptRow] from [sysobjects] SO where xtype = 'U'

union -- Add in the closing part
select ' ' as [Script], SO.[Name] as [Table_Name], 2002 as [ScriptRow] from [sysobjects] SO where xtype = 'U'

)combined
--where Table_Name like 'air%'
order by Table_Name, ScriptRow

-----
-- ** Update for formulas for computed columns
-----

update ST
set Script = F.Script
from
@ScriptTable ST inner join
--select Script, TableName, ScriptRow from
(select space(30) +
case when [Column_ID] = 1 then ' ' else ',' end +
 '[' + ac.name + '] AS ' + sc.text as Script,
so.name as Table_Name,
sc.number + 1000 as ScriptRow
from
sysobjects so left join syscomments sc on so.id = sc.id
left join sys.all_columns ac on sc.id = ac.[object_id] and sc.number = ac.[column_id]
where
so.xtype = 'U' -- Is user defined table
and
sc.number is not null -- number is the column number
) F
on ST.Table_Name = F.Table_Name and ST.ScriptRow = F.ScriptRow

-----
-- ** Add the index create statements (3000)
-----

declare @ScriptText varchar(7500)
declare @TableNameText varchar(255)
declare @ScriptRowInt int

declare @i int
declare @keyCols varchar(2000)

declare @ScriptCursor CURSOR

set nocount ON

insert into @ScriptTable
select
'CREATE ' + UniqueText + ClusterText + ' INDEX [' + [Index_Name] + '] ON [DBO].['+[Table_Name]+' ]' -- (' + TargetField
as [Script],
Table_Name as [Table_Name],

```

```

3000 + (indexID*2) as [ScriptRow]
from(
select
i.index_id as IndexID,
i.name as Index_Name,
case when i.type_desc = 'CLUSTERED' then 'CLUSTERED' else 'NONCLUSTERED' end as ClusterText,
case when i.is_unique = 1 then 'UNIQUE ' else '' end as UniqueText,
s.[object_id] as Table_ID,
so.name as Table_Name
from [sys].[indexes] I
left join [sys].[stats] S on I.[object_id] = S.[object_id] and I.[index_id] = S.[stats_id]
left join [sysobjects] SO on S.[object_id] = SO.[id]
--left join [sys].[all_columns] AC on S.[object_id] = AC.[object_id]
where so.xtype = 'U'
)inside
WHERE Table_Name in (Select distinct Table_Name from @ScriptTable)
order by [Table_Name], [ScriptRow]

SET @ScriptCursor = CURSOR
FOR
select [Script], [Table_Name], [ScriptRow]
from @ScriptTable
where ScriptRow > 3000 and ScriptRow < 4000
FOR UPDATE

set nocount ON

OPEN @ScriptCursor
FETCH NEXT FROM @ScriptCursor INTO @ScriptText, @TableNameText, @ScriptRowInt
WHILE @@FETCH_STATUS = 0
BEGIN
set @keyCols = ''
set @i = 1
WHILE index_col( @TableNameText, (@ScriptRowInt - 3000)/2, @i) IS NOT NULL
BEGIN
if @i > 1 begin set @keyCols = @keyCols + ',' end
set @keyCols = @keyCols + index_col( @TableNameText, (@ScriptRowInt - 3000)/2, @i)
set @i = @i + 1
END
UPDATE @ScriptTable SET [Script] = @ScriptText + ' (' + @keyCols + ')' WHERE CURRENT OF @ScriptCursor
--PRINT '--' + @ScriptText + ':' + @TableNameText + ':' + @keyCols -- + @ScriptRowInt
FETCH NEXT FROM @ScriptCursor INTO @ScriptText, @TableNameText, @ScriptRowInt
END
DEALLOCATE @ScriptCursor

-- ** Insert the 'GO' Statements afer each index create
insert into @ScriptTable
select '' as Script, Table_Name, ScriptRow + 1 as ScriptRow from @ScriptTable where ScriptRow > 3000 and ScriptRow < 4000 and
Script like 'CREATE%'

-----
-- ** Add the DEFAULT constraints (4000)
-----

insert into @ScriptTable
select
' ADD CONSTRAINT [' + DC.[Name] + '] DEFAULT ' + DC.[definition] + ' FOR [' + col_name(SO.[parent_obj], [parent_column_id]) + ']' as Script,
PSO.[Name] as Table_Name,
([parent_column_id]*3) + 4000 as ScriptRow
from sys.default_constraints DC
left join sysobjects SO on DC.object_id = SO.id
left join sysobjects PSO on SO.[parent_obj] = PSO.id

```

```

where
PS0. [Name] in (Select distinct Table_Name from @ScriptTable)

-- ** Insert the 'GO' Statements afer each constraint create
insert into @ScriptTable
select ' ' as Script, Table_Name, ScriptRow + 1 as ScriptRow from @ScriptTable where ScriptRow > 4000 and ScriptRow < 5000 and
Script like ' ADD CONSTRAINT%'

-- ** Insert the 'ALTER TABLE' part of the statement before each constraint
insert into @ScriptTable
select 'ALTER TABLE [DBO]. [' +Table_Name+']' as Script, Table_Name, ScriptRow - 1 as ScriptRow from @ScriptTable where ScriptRow >
4000 and ScriptRow < 5000 and Script like ' ADD CONSTRAINT%'

-----
-- ** Add the FOREIGN CONSTRAINTS
-----

declare @fkcursor cursor
declare @fkeyid int, @rkeyid int, @cnstid int, @fkeycol smallint, @rkeycol smallint, @fkeycounter int
declare @parentTbl varchar(300), @referenceTbl varchar(300), @keyName varchar(300), @keys varchar(500), @cnstdes varchar(500)

set @fkeycounter = 5001

set @fkcursor = cursor for
select
FK. [object_id], FK. parent_object_id, FK. referenced_object_id,
S0. name as KeyName, PS0. name as ParentTable, RS0. name as ReferenceTable
from
sys. [foreign_keys] FK
left join sysobjects S0 on FK. object_id = S0. id
left join sysobjects PS0 on FK. [parent_object_id] = PS0. id
left join sysobjects RS0 on FK. [referenced_object_id] = RS0. id
WHERE PS0. name in (Select distinct Table_Name from @ScriptTable)

open @fkcursor
fetch @fkcursor into @cnstid, @fkeyid, @rkeyid, @keyName, @parentTbl, @referenceTbl
WHILE @@FETCH_STATUS = 0
Begin
declare ms_crs_fkey cursor local for
select parent_column_id, referenced_column_id from sys. foreign_key_columns where constraint_object_id = @cnstid
open ms_crs_fkey
fetch ms_crs_fkey into @fkeycol, @rkeycol
select @keys = col_name(@fkeyid, @fkeycol), @cnstdes = col_name(@rkeyid, @rkeycol)
fetch ms_crs_fkey into @fkeycol, @rkeycol
while @@fetch_status >= 0
begin
select @keys = @keys + ', ' + col_name(@fkeyid, @fkeycol),
@cnstdes = @cnstdes + ', ' + col_name(@rkeyid, @rkeycol)
fetch ms_crs_fkey into @fkeycol, @rkeycol
end

insert into @ScriptTable(Script, Table_Name, ScriptRow)
values(' ALTER TABLE DBO. [' +@parentTbl +'] ADD CONSTRAINT [' +@keyName+'],' ,@parentTbl ,@fkeycounter)

insert into @ScriptTable(Script, Table_Name, ScriptRow)
values(' FOREIGN KEY (' + @keys + ') REFERENCES DBO. [' + @referenceTbl +'] (' + @cnstdes +'),' ,@parentTbl ,@fkeycounter+1)

insert into @ScriptTable(Script, Table_Name, ScriptRow)
values(' ', @parentTbl ,@fkeycounter+2)

deallocate ms_crs_fkey
set @fkeycounter = @fkeycounter + 6

```

```

Fetch next from @fkcursor into @cnstid, @fkeyid, @rkeyid, @keyName, @parentTbl, @referenceTbl
end

-- ** Insert the 'A comment before each table create and space after'
insert into @ScriptTable
select distinct
'/* [' + Table_Name + ' ] */' as Script,
Table_Name, 900 as ScriptRow from @ScriptTable

insert into @ScriptTable
select distinct
' ' as Script,
Table_Name, 9000 as ScriptRow from @ScriptTable

-- Clean the Results
delete from @scripttable where table_name like 'sys%' or table_name = 'dtproperties'

-- Parse the code into the final table
declare @tablesScript TABLE (table_name varchar(255), script varchar(7500))

declare @finalCursor CURSOR

declare @builddupText varchar(7500)
set @builddupText = ''

set @finalCursor = cursor for
select Script, Table_Name, ScriptRow from @scripttable order by table_name, scriptrow

OPEN @finalCursor
FETCH NEXT FROM @finalCursor INTO @ScriptText, @TableNameText, @ScriptRowInt
WHILE @@FETCH_STATUS = 0
begin
if (@ScriptRowInt = 9000)
begin
insert into @tablesScript (table_name, script) values (@TableNameText, @BuilddupText + char(13) + @ScriptText )
set @builddupText = ''
end
else
begin
set @builddupText = @builddupText + char(13) + @ScriptText
end
FETCH NEXT FROM @finalCursor INTO @ScriptText, @TableNameText, @ScriptRowInt
END
deallocate @finalcursor

-- Contents of table script are now located in the @tablesScript temporary table
/*****
END TABLE SCRIPT CREATION (this footer is repeated at the beginning)
*****/

/*****
BEGIN PROGRAMABILITY SCRIPT CREATION (this footer is repeated at the end)
*****/

declare @procsCursor CURSOR

set @procsCursor = CURSOR FOR
select distinct
upper(SysObjects.Name) as ObjectName
from SysObjects
where (SysObjects.Category = 0) and

```

```
sysobjects.type in ('P','FN','TF','V') and --Only want procs, functions and views  
not (upper(left([SysObjects].[Name],3)) in ('SP_','DT_','XP_','ZZZ','FN_'))
```

```
declare @objectName varchar(255)  
declare @OutStringP varchar(MAX)  
declare @InStringP varchar(255)  
declare @TempCursorP CURSOR
```

```
declare @TempTableP TABLE  
( [text] varchar(7500) )
```

```
declare @programScript TABLE ([object_name] varchar(255), script varchar(max) ) -- varchar(8000))
```

```
declare @scriptString varchar(max)  
set @scriptString = ''
```

```
declare @tlen float
```

```
INSERT into @programScript([object_name], script) values(@objectName, '')
```

```
OPEN @procsCursor FETCH NEXT FROM @procsCursor into @objectName
```

```
WHILE @@FETCH_STATUS = 0
```

```
BEGIN
```

```
set @tlen = 0
```

```
set @OutStringP = ''
```

```
set nocount ON
```

```
insert into @TempTableP exec sp_helptext @objectName
```

```
SET @TempCursorP = CURSOR FAST_FORWARD FOR select [Text] from @TempTableP
```

```
OPEN @TempCursorP FETCH NEXT FROM @TempCursorP INTO @InStringP
```

```
WHILE @@FETCH_STATUS = 0
```

```
BEGIN
```

```
IF(len(@InStringP) < 255)
```

```
BEGIN
```

```
IF(LEN(LTRIM(RTRIM(@OutStringP + @InStringP))) > 0)
```

```
BEGIN
```

```
set @scriptString = @scriptString + CAST(@OutStringP AS varchar(max)) + CAST(@InStringP AS varchar(max))
```

```
set @tlen = @tlen + len(@outStringP + @InStringP)
```

```
--UPDATE @programScript set [script] = [script] + @OutString + @Instring where [object_name] = @objectName
```

```
--print RTRIM(@OutString + @InString)
```

```
END
```

```
set @OutStringP = ''
```

```
END
```

```
ELSE
```

```
BEGIN
```

```
set @OutStringP = @OutStringP + @InStringP
```

```
END
```

```
FETCH NEXT FROM @TempCursorP INTO @InStringP
```

```
END
```

```
INSERT into @programScript([object_name], script) values(@objectName, @scriptString)
```

```
set @scriptString = ''
```

```
CLOSE @TempCursorP
```

```
DEALLOCATE @TempCursorP
```

```
DELETE FROM @TempTableP
FETCH NEXT FROM @procsCursor INTO @objectName
END
```

```
-- Contents of procs/views/functions script are now located in the @programScript temporary table
/*****
END PROGRAMABILITY SCRIPT CREATION (this footer is repeated at the beginning)
*****/
```

```
set nocount on
```

```
declare @ProcName nvarchar(100)
declare @ProcSortOrder varchar(30)
declare @MyCursor CURSOR
declare @ObjectType char(2)
declare @TypeString varchar(20)
declare @CheckParm varchar(20)
```

```
declare @ProcedureCode varchar(7500)
```

```
declare @TempCursor CURSOR
declare @InString varchar(7500)
declare @OutString varchar(7500)
declare @TempTable TABLE
( [text] varchar(7500) )
```

```
declare @StoredProcs TABLE
(
SortOrder int,
ProcedureName varchar(255),
ProcedureCode varchar(max),
ObjectType char(2),
TypeString varchar(20),
CheckParm varchar(20),
UsedBy int,
Uses int
)
```

```
set nocount ON
Insert Into @StoredProcs
select distinct
case SysObjects.type
when 'FN' then 1 --IsScalarFunction
when 'TF' then 2 --IsTableFunction
when 'V' then 5 --IsView
when 'U' then 4 --IsTable
ELSE 3 --IsProcedure
end as SortOrder,
```

```
upper(SysObjects.Name),
case SysObjects.type
when 'U' then TS.script
else PS.script -- SysComments.Text
END as ProcedureCode,
SysObjects.type,
case SysObjects.type
when 'P' then 'Procedure'
when 'V' then 'View'
when 'U' then 'Table'
ELSE 'Function'
```

```

END as TypeString,
case SysObjects.type
when 'FN' then 'IsScalarFunction'
when 'TF' then 'IsTableFunction'
when 'V' then 'IsView'
when 'U' then 'IsTable'
ELSE 'IsProcedure'
END as CheckParm,
0 as UsedBy,
0 as Uses

from SysObjects --left join SysComments on SysObjects.ID = SysComments.ID
left join @tablesScript TS on SysObjects.Name = TS.table_name
left join @programScript PS on SysObjects.Name = PS.[object_name]
where
SysObjects.type in ('P', 'FN', 'TF', 'V', 'U') and --Only want procs, functions and views
(SysObjects.Category = 0)
and not (upper(left([SysObjects].[Name], 3)) in ('SP_', 'DT_', 'XP_', 'ZZZ', 'FN_')) --exclude system and ZZZ hidden stuff
--order by SysObjects.Name ASC

--select count(sortorder), procedureName from @StoredProcs group by procedureName order by ProcedureName

--create table ##SPuses
declare @SPuses TABLE
(
[ObjectName] varchar(255),
UsesName varchar(255),
Lvl int,
Ord int
)

declare @level int
set @level = 1

declare @rowsAffected int
set @rowsAffected = 1000

--insert into @spuses
insert into @SPuses
select
A2.ProcedureName as [ObjectName],
B2.ProcedureName as [UsesName],
1 as Lvl,
0 as Ord
from @StoredProcs A2 left join @StoredProcs B2
on (REPLACE(UPPER(A2.ProcedureCode), A2.ProcedureName, ''))
LIKE '%' + upper(B2.ProcedureName) + '%'
order by A2.SortOrder DESC

update SP set
UsedBy = coalesce(SPAg.UsedByCount, 0),
Uses = coalesce(SPAg2.UsesCount, 0)
from @StoredProcs SP
left join
(select count(ObjectName) as UsedByCount, UsesName from @SPuses --@spuses
group by UsesName) SPAg
on SP.ProcedureName = SPAg.usesName
left join
(select count(UsesName) as UsesCount, ObjectName from @SPuses --@spuses
where UsesName is not null group by ObjectName) SPAg2
on SP.ProcedureName = SPAg2.ObjectName

```

```

while @rowsAffected > 0
begin
insert into @SPuses
select distinct
SP1.ObjectName as [ObjectName],
SP2.UsesName as [UsesName],
@level + 1 as Lvl,
0 as Ord
from
@SPuses SP1 left join @SPuses SP2 on SP1.UsesName = SP2.ObjectName
where SP1.UsesName is not null and SP2.UsesName is not null
and
SP1.Lvl = @level

set @rowsAffected = @@rowcount
--print @rowsAffected

set @level = @level + 1
end

insert into @SPuses
select distinct ObjectName, UsesName, 0 as Lvl, 1 as Ord
from @SPuses order by ObjectName

delete from @SPuses where Ord = 0

update @SPuses set Ord = 0

set @level = 1
update @SPuses set Ord = @level where usesName is null
set @rowsAffected = @@rowcount
--print @rowsAffected

while @rowsAffected > 0
begin
update @SPuses set Ord = @level + 1 where usesName in (select ObjectName from @SPuses where Ord = @level)

set @rowsAffected = @@rowcount
-- print @rowsAffected

set @level = @level + 1
end

set nocount ON

SET @MyCursor = CURSOR SCROLL --FAST_FORWARD
FOR
select ProcedureName,
cast(outC.SortOrder as varchar(2)) + '.' + cast(outC.MxOrd as varchar(2)) + '.' + cast(outC.AvOrd as varchar(2)) + '.' + cast(outC.MnOrd
as varchar(2)) as SortOrder
, ObjectType
, TypeString
, CheckParm
, ProcedureCode
from
(
select distinct ProcedureName,
SortOrder
, ObjectType

```

```

,@TypeString
,@CheckParm
,@ProcedureCode
,B. MxOrd, B. AvOrd, B. MnOrd
from @StoredProcs A

left join (select max(ord) as MxOrd, Avg(ord) as AvOrd, min(ord) as MnOrd, ObjectName from @SPuses group by ObjectName ) B
on A.ProcedureName = B.ObjectName
) outC
order by outC.SortOrder, outC.MxOrd, outC.AvOrd, outC.MnOrd

--select max(ord) as MxOrd, Avg(ord) as AvOrd, min(ord) as MnOrd, ObjectName from @SPuses group by ObjectName order by MxOrd, AvOrd,
MnOrd

-----
set nocount ON

print '-----'
Print '-- The procs/functions and views will be generated in the following order [Rank]'
print ' '
OPEN @MyCursor
FETCH NEXT FROM @MyCursor INTO @ProcName, @ProcSortOrder, @ObjectType, @TypeString, @CheckParm, @ProcedureCode
WHILE @@FETCH_STATUS = 0
BEGIN
PRINT '--' + @ObjectType + ' : ' + left(@TypeString+' ', 10) + ': ' + @ProcName + ' [' + cast(@ProcSortOrder as varchar(20)) + ']'
FETCH NEXT FROM @MyCursor INTO @ProcName, @ProcSortOrder, @ObjectType, @TypeString, @CheckParm, @ProcedureCode
END
set nocount ON

FETCH FIRST FROM @MyCursor
INTO @ProcName, @ProcSortOrder, @ObjectType, @TypeString, @CheckParm, @ProcedureCode
WHILE @@FETCH_STATUS = 0
BEGIN
print '-----'
--PRINT 'if exists (select * from dbo.sysobjects '
--PRINT ' where id = object_id(N' + char(39) + '[dbo].[' + @ProcName + ']' + char(39) + '))'
--PRINT ' and OBJECTPROPERTY(id, N' + char(39) + '@CheckParm + char(39) + ') = 1) '
-- PRINT ' drop ' + @TypeString+' ' + @ProcName
-- PRINT ' GO '
-- PRINT ' SET QUOTED_IDENTIFIER OFF '
-- PRINT ' GO '
-- PRINT ' SET ANSI_NULLS OFF '
-- PRINT ' GO'

/*if @TypeString = 'Function' or @TypeString = 'Procedure' or @TypeString = 'View'
begin

end*/

if @TypeString = 'Table'
begin
print @ProcedureCode
end
else
begin
print 'GO'
-- exec sp_helptext @ProcName
-- The following section compensates for the way sp_helptext handles wrapping
-----
set @OutString = ''

```

```

set nocount ON
insert into @TempTable exec sp_helptext @ProcName
SET @TempCursor = CURSOR FAST_FORWARD FOR select [Text] from @TempTable

OPEN @TempCursor FETCH NEXT FROM @TempCursor INTO @InString
WHILE @@FETCH_STATUS = 0
BEGIN
IF(LEN(@InString) < 255)
BEGIN
IF(LEN(LTRIM(RTRIM(@OutString + @InString))) > 0)
BEGIN
print RTRIM(@OutString + @InString)
END
set @OutString = ''
END
ELSE
BEGIN
set @OutString = @OutString + @InString
END

FETCH NEXT FROM @TempCursor INTO @InString
END

CLOSE @TempCursor
DEALLOCATE @TempCursor
DELETE FROM @TempTable

--if @TypeString = 'Function' or @TypeString = 'Procedure' or @TypeString = 'View'
--begin
print 'GO'
--end

-----
end
-- PRINT ' GO '
-- PRINT ' SET QUOTED_IDENTIFIER OFF '
-- PRINT ' GO '
-- PRINT ' SET ANSI_NULLS ON '
-- PRINT ' GO '
-- PRINT ''
-- PRINT ' GRANT EXECUTE ON [dbo].[' + @ProcName + '] TO [' + @ProcUser + ']'
-- PRINT ' GO '

FETCH NEXT FROM @MyCursor

INTO @ProcName, @ProcSortOrder, @ObjectType, @TypeString, @CheckParm, @ProcedureCode
END

CLOSE @MyCursor
DEALLOCATE @MyCursor

```