



Refactor Roadmap

Item #	Refactor name	Full Description	Time estimate to complete (Man days)	Comments	Status	ApexSQL Refactor	ApexSQL Edit
1	Qualify Object Names	Qualify Object Names refactoring goal is to add missing scheme name to object names and datasource name (tables, views, etc) to column names.	18		Approved for Production	2010.01 or higher	2008.05 or higher
2	Expand Wildcards	All * wildcards are replaced with corresponding list of columns. If it is not possible to identify list of columns using metadata then wildcard remains unchanged.	8		Approved for Production	2010.01 or higher	2008.05 or higher
3	Generate Client Code	Modifies selected script to an equivalent programming language (i.e. C#, VB.NET, PHP) and saves the changes to the clipboard.	6		Approved for Production	2010.01 or higher	2008.06 or higher
4	SQL syntax error highlighting	Checks input script for lexical, syntax, and semantic errors.	6		Approved for Production		2008.06 or higher
5	Encapsulate As Scalar Function	Scalar expression (or part of the expression) is extracted into new scalar function. Expression part to extract is highlighted by user. Extracted expression fragment is replaced with newly created function call.	6	Related with Encapsulate (6, 7, 14)	Approved for Production	2010.01 or higher	2008.06 or higher
6	Encapsulate As Inline Table-Valued Function	Query to extract is highlighted by user. Highlighted statement is moved into new table function and replaced with a newly created function call. Query that is extracted into the table function can contain parameters.	6	Related with Encapsulate (5, 7, 14)	Approved for Production	2010.01 or higher	2008.08 or higher
7	Encapsulate As Stored Procedure	New procedure is created with a code fragment highlighted by user. Highlighted code is replaced with a newly created procedure call. All required parameters are passed into new procedure.	6	Related with Encapsulate (5, 6, 14)	Approved for Production	2010.01 or higher	2008.08 or higher
8	Smart Rename columns	Column with the new name is inserted in to the table. All data is copied into this column from old column and all column usage is updated to use new name. Old column is dropped. This refactoring is applied to Tables and Views	8		Approved for Production		2008.10 or higher



9	Remove parameter	Parameter is removed from function or procedure declaration and all calls. Procedure/function body may still contain removed parameter usage (it is not possible to remove it automatically). This refactoring is applied to Functions and Procedures	4	Related with change signature refactorings (9, 15, 16)	Approved for Production		2008.10 or higher
10	Smart Rename parameters	ALTER statement is created that modifies function/procedure declaration and all parameter's usage. This refactoring is applied to Functions and Procedures	6		Approved for Production		
11	Adding surrogate key	New column will be added to the source table and filled with unique values. Foreign key column will be added to all linked tables too and filled with corresponding values. This refactoring is applied to Tables	10		Approved for Production		
12	Smart Rename	To rename object it is required to use sp_rename or to create new object with necessary name (if sp_rename usage is not possible). After that all old object usage is replaced with new object name (including functions and procedures code). This refactoring is applied to Tables, Views, Functions and Procedures	36		Approved for Production		
13	Replace one-to-many link with associative table	To introduce this refactoring new associative table should be created and filled with source tables' data. Also it is required to update all queries that use source tables joins.	8		Approved for Production		
14	Encapsulate As View	Highlighted query is extracted into new view and replaced with a newly created view usage. Query extracted into view can't have parameters (this is a key distinction from Encapsulate As Inline Table-Valued Function refactoring).	6	Related with Encapsulate (5, 6, 7)	Approved for Production		
15	Introduce parameter	New parameter is added to function or procedure declaration. All procedure or function calls are updated to use additional new parameter (value is set by user).	4	Related with change signature refactorings (9, 15, 16)	Approved for Production		
16	Change signature	Function/procedure declaration is modified to correspond with required order. All procedure/function calls are updated accordingly too.	4	Related with change signature refactorings (9, 15, 16)	Approved for Production		
17	Introduce variable	To execute this refactoring it is required to create new variable in the procedure/function code and assign highlighted expression to this variable. Newly created variable will replace highlighted expression.	4		Approved for Production		



18	Find Unused Variables and Parameters	Script is searched for unused variables and parameters. All found instances are highlighted (by underline for example).	4		Approved for Production		
19	Split Table	To execute this refactoring new table with extracted columns is created. Data from the source table is transferred into newly created table. Unnecessary columns are dropped in the source table. Also all source table usages are updated to use new table when required.	24		Not Started		
20	Introduce calculatable column	New column should be added to the source table (column name and type is entered by the user). To calculate new column value function template is defined.	6		Not Started		
21	Find usage {advanced}	Source script is analyzed for selected object usage (object can be selected from the list or by placing cursor on the object name). All usages found are highlighted (by color for example). Also all usages can be shown in a separate list.	14	Related with Rename (8, 10, 12)	Not Started		
22	Split column	To execute refactoring it is required to create new columns and fill newly created columns with data. To fill newly created columns masks or regular expression can be used to extract data from source field. All source column usages should be updated to one of the newly created columns (it is not possible to do that automatically, so some interface should be provided). Source column can be dropped once refactoring is complete.	6	Related with Find usage (21)	Not Started		
23	Merge Tables	New table is created that contain all old tables' columns. Data is transferred from source tables into newly created table. All tables' usage is updated to use new table.	18		Not Started		
24	Moving of column	New column is created in the target table and all data is copied from the source column. All source column usages are replaced with newly created column. Source column is dropped from the table.	8	Related with Find usage (21)	Not Started		
25	"Move Schema" Refactoring	ALTER SCHEMA should be executed to transfer object into new schema. After that all object usage is updated to use new object name (including functions and procedures)	8	Implemented in Rename (12)	Approved for Production		
26	Replacing auxiliary key with natural	Identify columns that will be used for primary keys. Columns containing primary key should have unique data	10		Not Started		



	key	for each record. New index is added for natural key. Linked tables are updated to used natural key. Surrogate key is removed from source table.					
27	Merge columns	New column is created and filled with a data from source columns. All source columns usages are updated with a new column usage. Source columns can be dropped once refactoring is complete.	6	Related with Find usage (21)	Not Started		
28	Parameterization of method	New procedure containing all source procedure's parameters should be created. One additional parameter will be added to distinguish code branches. Type of the new parameter and its value for every branch is set by user. New procedure body consists of source procedure's code. All source procedure's calls are replaced with newly created procedure. After refactoring source procedures can be dropped.	10		Not Started		
29	Introduce default value	New DEFAULT object should be created and linked to the source column	6		Not Started		
30	Inline of Stored Procedure	Drops selected procedure and replaces all calls with a procedure body.	6	Related with Inline (30, 31, 32, 33)	Not Started		
31	Inline of Scalar Function	Drops selected function and replaces all calls with a function body.	6	Related with Inline (30, 31, 32, 33)	Not Started		
32	Inline of Inline Table-Valued Function	Drops selected function and replaces all calls with a function body.	6	Related with (30, 31, 32, 33)	Not Started		
33	Inline of View	Removes selected view and replaces all usages with a view query.	6	Related with Inline (30, 31, 32, 33)	Not Started		
34	Introduce trigger to collect historical data	Table is created to store historical data. To fill newly created table on data changes trigger is added.	6		Not Started		
35	Transform column to allow NULL	Source column should be altered to allow NULL value. All default values for this column should be replaced with NULL.	6		Not Started		

Examples:

Example #1: Qualify Object Names

Before refactoring



```
SELECT EmployeeID
FROM Employee p1
GROUP BY EmployeeID
HAVING MAX(ContactID) >= ALL
    (SELECT 2 * AVG(ContactID)
     FROM Contact p2
     WHERE EmployeeID = ContactID) ;
```

After refactoring

```
SELECT p1.EmployeeID
FROM dbo.Employee p1
GROUP BY p1.EmployeeID
HAVING MAX(p1.ContactID) >= ALL
    (SELECT 2 * AVG(p2.ContactID)
     FROM dbo.Contact p2
     WHERE p1.EmployeeID = p2.ContactID) ;
```

Example #2: Expand Wildcards

Script before refactoring

```
WITH DirReps(EmployeeID, DirectReports) AS
(
    SELECT EmployeeID, COUNT(EmployeeID)
    FROM Employee AS e
    WHERE EmployeeID IS NOT NULL
    GROUP BY EmployeeID
)
SELECT * FROM DirReps ORDER BY EmployeeID;
```

Script after refactoring

```
WITH DirReps(EmployeeID, DirectReports) AS
(
    SELECT EmployeeID, COUNT(EmployeeID)
    FROM Employee AS e
    WHERE EmployeeID IS NOT NULL
    GROUP BY EmployeeID
)
SELECT DirReps.EmployeeID, DirReps.DirectReports FROM DirReps ORDER BY EmployeeID;
```

Example #3: Generate Client Code

SQL Script

```
SELECT *
FROM ICE_Contacts
```

Generated Code

```
Dim SQL As String = _
    "SELECT *" + vbCrLf + _
    " FROM ICE_Contacts"
```

Example #4: SQL Syntax Error Highlighting

Original Script with Errors

```
ALTER TABLE [dbo].[ErrorLog]
    ADD [User] [sysname] COLLATES Latin1_General_CS_AS NOT NULL;
GO
UPDATE [dbo].[ErrorLog]
```



```
SET
 [User] = [UserName];
GO
```

Script with errors highlighted

```
ALTER TABLE [dbo].[ErrorLog]
  ADD [User] [sysname] COLLATES Latin1_General_CS_AS NOT NULL;
GO
UPDATE [dbo].[ErrorLog]
SET
  [User] = [UserName];
GO
```

Example #5: Encapsulate As Scalar Function

Object script

```
CREATE PROCEDURE Procedure1 as
Return select log(exp(object_id%10) + sin(object_id)), * from sys.objects
```

Generated script

```
CREATE FUNCTION Calculate (@value int)
returns float
as
begin
  return log(exp(@value%10) + sin(@value))
end

ALTER PROCEDURE Procedure1 as
Return select dbo.Calculate(object_id), * from sys.objects
```

Example #6: Encapsulate As Inline Table-Valued Function

Query is extracted into separate GetSalesPersonPerform function.

Object Script

```
CREATE PROCEDURE Procedure1(@topCount int)
as
SELECT * FROM Sales.SalesPerson s
WHERE s.SalesPersonID in
  (SELECT SalesPersonID FROM
   (SELECT TOP(@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
   FROM Sales.SalesOrderHeader
   GROUP BY SalesPersonID
   ORDER BY TotalSales DESC) TopSalesPerson
  )
GO
```

Generated script

```
CREATE Function GetSalesPersonPerform(@topCount int)
RETURNS TABLE
AS
RETURN SELECT TOP(@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
ORDER BY TotalSales DESC
GO
ALTER PROCEDURE Procedure1(@topCount int)
```



```
as
SELECT * FROM Sales.SalesPerson s
WHERE s.SalesPersonID in
      (SELECT SalesPersonID FROM dbo.GetSalesPersonPerform(@topCount) TopSalesPerson)
GO
```

Example #7: Encapsulate As Stored Procedure

New uspPrintError procedure extraction from uspLogError procedure.

Object Script

```
-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
CREATE PROCEDURE dbo.uspLogError
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;

        -- Return if inside an uncommittable transaction.
        -- Data insertion/modification is not allowed when
        -- a transaction is in an uncommittable state.
        IF XACT_STATE() = -1
        BEGIN
            PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
            + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
            RETURN;
        END

        INSERT dbo.NewErrorLog
        (
            [UserName],
            [ErrorNumber],
            [ErrorSeverity],
            [ErrorState],
            [ErrorProcedure],
            [ErrorLine],
            [ErrorMessage]
        )
        VALUES
        (
            CONVERT(sysname, CURRENT_USER),
            ERROR_NUMBER(),
            ERROR_SEVERITY(),
            ERROR_STATE(),
            ERROR_PROCEDURE(),
            ERROR_LINE(),
            ERROR_MESSAGE()
        );

        -- Pass back the ErrorLogID of the row inserted
        SET @ErrorLogID = @@IDENTITY;
    END TRY
```



```
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    SET NOCOUNT ON;

    -- Print error information.
    PRINT 'Error ' + CONVERT(varchar(50), ERROR_NUMBER()) +
        ', Severity ' + CONVERT(varchar(5), ERROR_SEVERITY()) +
        ', State ' + CONVERT(varchar(5), ERROR_STATE()) +
        ', Procedure ' + ISNULL(ERROR_PROCEDURE(), '-') +
        ', Line ' + CONVERT(varchar(5), ERROR_LINE());
    PRINT ERROR_MESSAGE();

    RETURN -1;
END CATCH
END;
```

Generated script

```
-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
ALTER PROCEDURE dbo.uspLogError
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
    AS -- by uspLogError in the ErrorLog table
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;

        -- Return if inside an uncommittable transaction.
        -- Data insertion/modification is not allowed when
        -- a transaction is in an uncommittable state.
        IF XACT_STATE() = -1
            BEGIN
                PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
                    + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
                RETURN;
            END
    END

    INSERT dbo.NewErrorLog
        (
            [UserName],
            [ErrorNumber],
            [ErrorSeverity],
            [ErrorState],
            [ErrorProcedure],
            [ErrorLine],
            [ErrorMessage]
        )
    VALUES
        (
            CONVERT(sysname, CURRENT_USER),
            ERROR_NUMBER(),
            ERROR_SEVERITY(),
            ERROR_STATE(),
            ERROR_PROCEDURE(),
            ERROR_LINE(),
            ERROR_MESSAGE()
        );
END;
```



```
-- Pass back the ErrorLogID of the row inserted
SET @ErrorLogID = @@IDENTITY;
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;

CREATE PROCEDURE [dbo].[uspPrintError]
AS
BEGIN
    SET NOCOUNT ON;

    -- Print error information.
    PRINT 'Error ' + CONVERT(varchar(50), ERROR_NUMBER()) +
        ', Severity ' + CONVERT(varchar(5), ERROR_SEVERITY()) +
        ', State ' + CONVERT(varchar(5), ERROR_STATE()) +
        ', Procedure ' + ISNULL(ERROR_PROCEDURE(), '-') +
        ', Line ' + CONVERT(varchar(5), ERROR_LINE());
    PRINT ERROR_MESSAGE();
END;
```

Example #8: Smart Rename columns

ErrorLog table UserName column is renamed into User.

Object Script

```
CREATE TABLE [dbo].[ErrorLog] (
    [ErrorLogID] [int] IDENTITY(1,1) NOT NULL,
    [ErrorTime] [datetime] NOT NULL CONSTRAINT [DF_ErrorLog_ErrorTime] DEFAULT (getdate()),
    [UserName] [sysname] COLLATE Latin1_General_CS_AS NOT NULL,
    [ErrorNumber] [int] NOT NULL,
    [ErrorSeverity] [int] NULL,
    [ErrorState] [int] NULL,
    [ErrorProcedure] [nvarchar] (126) COLLATE Latin1_General_CS_AS NULL,
    [ErrorLine] [int] NULL,
    [ErrorMessage] [nvarchar] (4000) COLLATE Latin1_General_CS_AS NOT NULL,
    CONSTRAINT [PK_ErrorLog_ErrorLogID] PRIMARY KEY CLUSTERED
)
    ([ErrorLogID] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
CREATE PROCEDURE [dbo].[uspLogError]
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;
    END TRY
```



```
-- Return if inside an uncommittable transaction.
-- Data insertion/modification is not allowed when
-- a transaction is in an uncommittable state.
IF XACT_STATE() = -1
BEGIN
    PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
        + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
    RETURN;
END

INSERT [dbo].[ErrorLog]
(
    [UserName],
    [ErrorNumber],
    [ErrorSeverity],
    [ErrorState],
    [ErrorProcedure],
    [ErrorLine],
    [ErrorMessage]
)
VALUES
(
    CONVERT(sysname, CURRENT_USER),
    ERROR_NUMBER(),
    ERROR_SEVERITY(),
    ERROR_STATE(),
    ERROR_PROCEDURE(),
    ERROR_LINE(),
    ERROR_MESSAGE()
);

-- Pass back the ErrorLogID of the row inserted
SET @ErrorLogID = @@IDENTITY;
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;
```

Generated script

```
ALTER TABLE [dbo].[ErrorLog]
    ADD [User] [sysname] COLLATE Latin1_General_CS_AS NOT NULL;

UPDATE [dbo].[ErrorLog]
SET
    [User] = [UserName];

ALTER TABLE [dbo].[ErrorLog]
    DROP COLUMN [UserName];

-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
ALTER PROCEDURE [dbo].[uspLogError]
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS
    -- by uspLogError in the ErrorLog table
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorID = 0;
```



```
BEGIN TRY
-- Return if there is no error information to log
IF ERROR_NUMBER() IS NULL
    RETURN;

-- Return if inside an uncommittable transaction.
-- Data insertion/modification is not allowed when
-- a transaction is in an uncommittable state.
IF XACT_STATE() = -1
BEGIN
    PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
    + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
    RETURN;
END

INSERT [dbo].[ErrorLog]
(
    [User],
    [ErrorNumber],
    [ErrorSeverity],
    [ErrorState],
    [ErrorProcedure],
    [ErrorLine],
    [ErrorMessage]
)
VALUES
(
    CONVERT(sysname, CURRENT_USER),
    ERROR_NUMBER(),
    ERROR_SEVERITY(),
    ERROR_STATE(),
    ERROR_PROCEDURE(),
    ERROR_LINE(),
    ERROR_MESSAGE()
);

-- Pass back the ErrorLogID of the row inserted
SET @ErrorID = @@IDENTITY;
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;
```

Example #9: Remove parameter

Object Script

```
CREATE Function [dbo].[GetSalesPersonPerform](@topCount int, @minSum money)
RETURNS TABLE
AS
RETURN SELECT TOP(@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
HAVING Sum(TotalDue)>0
ORDER BY TotalSales DESC

CREATE PROCEDURE Procedure1
AS
SELECT * FROM [dbo].[GetSalesPersonPerform](7, 0)
```

Generated script



```
ALTER Function [dbo].[GetSalesPersonPerform] (@topCount int)
RETURNS TABLE
AS
RETURN SELECT TOP (@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
HAVING Sum(TotalDue)>0
ORDER BY TotalSales DESC

ALTER PROCEDURE Procedure1
AS
SELECT * FROM [dbo].[GetSalesPersonPerform] (7)
```

Example #10: Smart Rename parameters

uspLogError procedure ErrorLogID parameter rename into ErrorId.

Object Script

```
-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
CREATE PROCEDURE [dbo].[uspLogError]
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS
    -- by uspLogError in the ErrorLog table
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;

        -- Return if inside an uncommittable transaction.
        -- Data insertion/modification is not allowed when
        -- a transaction is in an uncommittable state.
        IF XACT_STATE() = -1
            BEGIN
                PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
                + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
                RETURN;
            END
        END

        INSERT [dbo].[ErrorLog]
        (
            [UserName],
            [ErrorNumber],
            [ErrorSeverity],
            [ErrorState],
            [ErrorProcedure],
            [ErrorLine],
            [ErrorMessage]
        )
        VALUES
        (
            CONVERT(sysname, CURRENT_USER),
            ERROR_NUMBER(),
            ERROR_SEVERITY(),
            ERROR_STATE(),
            ERROR_PROCEDURE(),
            ERROR_LINE(),
            ERROR_MESSAGE()
        )
    END TRY
    BEGIN CATCH
        -- This block is executed if an error occurs in the TRY block above
        IF XACT_STATE() = -1
            ROLLBACK TRANSACTION;
        ELSE
            COMMIT TRANSACTION;

        -- Record the error in the ErrorLog table
        IF @@ERROR = 0
            SET @ErrorLogID = 0;
        ELSE
            SET @ErrorLogID = @@ERROR;
    END CATCH
END
```



```
);  
  
-- Pass back the ErrorLogID of the row inserted  
SET @ErrorLogID = @@IDENTITY;  
END TRY  
BEGIN CATCH  
    PRINT 'An error occurred in stored procedure uspLogError: '  
    EXECUTE [dbo].[uspPrintError];  
    RETURN -1;  
END CATCH  
END;
```

Generated script

```
-- uspLogError logs error information in the ErrorLog table about the  
-- error that caused execution to jump to the CATCH block of a  
-- TRY...CATCH construct. This should be executed from within the scope  
-- of a CATCH block otherwise it will return without inserting error  
-- information.  
ALTER PROCEDURE [dbo].[uspLogError]  
    @ErrorID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted  
AS  
BEGIN -- by uspLogError in the ErrorLog table  
    SET NOCOUNT ON;  
  
    -- Output parameter value of 0 indicates that error  
    -- information was not logged  
    SET @ErrorID = 0;  
  
    BEGIN TRY  
        -- Return if there is no error information to log  
        IF ERROR_NUMBER() IS NULL  
            RETURN;  
  
        -- Return if inside an uncommittable transaction.  
        -- Data insertion/modification is not allowed when  
        -- a transaction is in an uncommittable state.  
        IF XACT_STATE() = -1  
        BEGIN  
            PRINT 'Cannot log error since the current transaction is in an uncommittable state. '  
            + 'Rollback the transaction before executing uspLogError in order to successfully log error  
information.';  
            RETURN;  
        END  
  
        INSERT [dbo].[ErrorLog]  
        (  
            [UserName],  
            [ErrorNumber],  
            [ErrorSeverity],  
            [ErrorState],  
            [ErrorProcedure],  
            [ErrorLine],  
            [ErrorMessage]  
        )  
        VALUES  
        (  
            CONVERT(sysname, CURRENT_USER),  
            ERROR_NUMBER(),  
            ERROR_SEVERITY(),  
            ERROR_STATE(),  
            ERROR_PROCEDURE(),  
            ERROR_LINE(),  
            ERROR_MESSAGE()  
        );  
  
        -- Pass back the ErrorLogID of the row inserted  
        SET @ErrorID = @@IDENTITY;
```



```
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;
```

Example #11: Adding surrogate key

Surrogate key is created for Order table.

Object Script

```
CREATE TABLE [Order]
(
    CustomerNumber int,
    OrderDate datetime,
    StoreId int,
    ShipTo int,
    BillTo int,
    Total float,
    PRIMARY KEY (CustomerNumber, OrderDate, StoreId)
)

CREATE TABLE OrderItem
(
    CustomerNumber int,
    OrderDate datetime,
    StoreId int,
    OrderItemNumber int,
    NumberOrdered int,
    PRIMARY KEY (CustomerNumber, OrderDate, StoreId)
)
```

Generated script

```
ALTER TABLE [Order] ADD OrderPOID int;
ALTER TABLE OrderItem ADD OrderPOID int;

CREATE TABLE #TempTable
(
    OrderPOID int IDENTITY(1,1),
    CustomerNumber int,
    OrderDate datetime,
    StoreId int
)

INSERT INTO #TempTable (CustomerNumber, OrderDate, StoreId)
SELECT CustomerNumber, OrderDate, StoreId FROM [Order] GROUP BY CustomerNumber, OrderDate, StoreId

UPDATE [Order]
SET OrderPOID = (SELECT OrderPOID FROM #TempTable
                WHERE #TempTable.CustomerNumber=[Order].CustomerNumber AND
                    #TempTable.OrderDate=[Order].OrderDate AND
                    #TempTable.StoreId=[Order].StoreId)

UPDATE [OrderItem]
SET OrderPOID = (SELECT OrderPOID FROM #TempTable
                WHERE #TempTable.CustomerNumber=[Order].CustomerNumber AND
                    #TempTable.OrderDate=[Order].OrderDate AND
                    #TempTable.StoreId=[Order].StoreId)

DROP TABLE #TempTable

ALTER TABLE [Order] ALTER COLUMN OrderPOID IDENTITY(1,1);
```



```
ALTER TABLE OrderItem ALTER COLUMN OrderPOID int IDENTITY(1,1);
```

Example #12: Smart Rename

Object Script

```
CREATE TABLE [dbo].[ErrorLog] (
    [ErrorLogID] [int] IDENTITY(1,1) NOT NULL,
    [ErrorTime] [datetime] NOT NULL CONSTRAINT [DF_ErrorLog_ErrorTime] DEFAULT (getdate()),
    [UserName] [sysname] COLLATE Latin1_General_CS_AS NOT NULL,
    [ErrorNumber] [int] NOT NULL,
    [ErrorSeverity] [int] NULL,
    [ErrorState] [int] NULL,
    [ErrorProcedure] [nvarchar](126) COLLATE Latin1_General_CS_AS NULL,
    [ErrorLine] [int] NULL,
    [ErrorMessage] [nvarchar](4000) COLLATE Latin1_General_CS_AS NOT NULL,
    CONSTRAINT [PK_ErrorLog_ErrorLogID] PRIMARY KEY CLUSTERED
)
    ([ErrorLogID] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]

-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
CREATE PROCEDURE [dbo].[uspLogError]
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS
    -- by uspLogError in the ErrorLog table
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;

        -- Return if inside an uncommittable transaction.
        -- Data insertion/modification is not allowed when
        -- a transaction is in an uncommittable state.
        IF XACT_STATE() = -1
            BEGIN
                PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
                    + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
                RETURN;
            END
    END TRY

    INSERT [dbo].[ErrorLog]
        (
            [UserName],
            [ErrorNumber],
            [ErrorSeverity],
            [ErrorState],
            [ErrorProcedure],
            [ErrorLine],
            [ErrorMessage]
        )
    VALUES
        (
            CONVERT(sysname, CURRENT_USER),
```



```
        ERROR_NUMBER(),
        ERROR_SEVERITY(),
        ERROR_STATE(),
        ERROR_PROCEDURE(),
        ERROR_LINE(),
        ERROR_MESSAGE()
    );

    -- Pass back the ErrorLogID of the row inserted
    SET @ErrorLogID = @@IDENTITY;
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;
```

Generated script

```
EXEC sp_rename N'[dbo].[ErrorLog]',N'NewErrorLog',N'OBJECT'

-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
ALTER PROCEDURE dbo.uspLogError
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;

        -- Return if inside an uncommittable transaction.
        -- Data insertion/modification is not allowed when
        -- a transaction is in an uncommittable state.
        IF XACT_STATE() = -1
            BEGIN
                PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
                    + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
                RETURN;
            END

        INSERT dbo.NewErrorLog
        (
            [UserName],
            [ErrorNumber],
            [ErrorSeverity],
            [ErrorState],
            [ErrorProcedure],
            [ErrorLine],
            [ErrorMessage]
        )
        VALUES
        (
            CONVERT(sysname, CURRENT_USER),
            ERROR_NUMBER(),
            ERROR_SEVERITY(),
```



```
        ERROR_STATE(),
        ERROR_PROCEDURE(),
        ERROR_LINE(),
        ERROR_MESSAGE()
    );

    -- Pass back the ErrorLogID of the row inserted
    SET @ErrorLogID = @@IDENTITY;
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;
```

Example #13: Replace one-to-many link with associative table

Object script

```
CREATE TABLE Customer
(
    CustomerPOID int PRIMARY KEY,
    Name varchar(100)
)

CREATE TABLE Policy
(
    PolicyId int PRIMARY KEY,
    CustomerPOID int FOREIGN KEY REFERENCES Customer(CustomerPOID),
    Amount money
)
```

Generated script

```
CREATE TABLE Holds
(
    CustomerPOID int,
    PolicyId int
)

INSERT INTO Holds(CustomerPOID, PolicyId)
SELECT CustomerPOID, PolicyId FROM Policy;

ALTER TABLE Customer DROP COLUMN CustomerPOID
```

Example #14: Encapsulate As View

Query is extracted into separate SalesPersonPerform view.

Object Script

```
CREATE PROCEDURE Procedure1
as
SELECT * FROM Sales.SalesPerson s
WHERE s.SalesPersonID in
    (SELECT SalesPersonID FROM
        (SELECT TOP 10 SalesPersonID, SUM(TotalDue) AS TotalSales
        FROM Sales.SalesOrderHeader
        GROUP BY SalesPersonID
        ORDER BY TotalSales DESC) TopSalesPerson
    )
GO
```



Generate script

```
CREATE VIEW SalesPersonPerform
AS
SELECT TOP 10 SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
ORDER BY TotalSales DESC

ALTER PROCEDURE Procedure1
as
SELECT * FROM Sales.SalesPerson s
WHERE s.SalesPersonID in
(SELECT SalesPersonID FROM SalesPersonPerform TopSalesPerson)
GO
```

Example #15: Introduce parameter

Object Script

```
CREATE Function [dbo].[GetSalesPersonPerform] (@topCount int)
RETURNS TABLE
AS
RETURN SELECT TOP (@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
HAVING Sum(TotalDue)>0
ORDER BY TotalSales DESC

CREATE PROCEDURE Procedure1
AS
SELECT * FROM [dbo].[GetSalesPersonPerform] (7)
```

Generated script

```
ALTER Function [dbo].[GetSalesPersonPerform] (@topCount int, @minSum money)
RETURNS TABLE
AS
RETURN SELECT TOP (@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
HAVING Sum(TotalDue)>0
ORDER BY TotalSales DESC

ALTER PROCEDURE Procedure1
AS
SELECT * FROM [dbo].[GetSalesPersonPerform] (7, 0)
```

Example #16: Introduce variable

@isBetweenDate variable extraction from (@inDate < @lastAccessDate AND @outDate > @lastAccessDate) expression. (bit type can be used for boolean expressions)

Object Script

```
CREATE FUNCTION DetermineAccountStatus
(
    @inAccountId float,
    @inDate datetime,
    @outDate datetime
)
RETURNS varchar
AS
BEGIN
    DECLARE @lastAccessDate datetime
```



```
SET @lastAccessDate = GetDate()

IF (@inDate < @lastAccessDate AND @outDate > @lastAccessDate) AND
(@inAccountId > 10000) AND
(@inAccountId <> 123456 AND @inAccountId <> 987654)
BEGIN
    PRINT 'Ok'
END

RETURN ''
END
```

Generated script

```
ALTER FUNCTION DetermineAccountStatus
(
    @inAccountId float,
    @inDate datetime,
    @outDate datetime
)
RETURNS varchar
AS
BEGIN
    DECLARE @lastAccessDate datetime
    SET @lastAccessDate = GetDate()

    DECLARE @isBetweenDate bit
    IF (@inDate < @lastAccessDate AND @outDate > @lastAccessDate)
        SET @isBetweenDate = 1
    ELSE
        SET @isBetweenDate = 0

    IF @isBetweenDate=1 AND
        (@inAccountId > 10000) AND
        (@inAccountId <> 123456 AND @inAccountId <> 987654)
    BEGIN
        PRINT 'Ok'
    END

    RETURN ''
END
```

Example #17: Find Unused Variables and Parameters

Script before refactoring

```
CREATE PROCEDURE [dbo].[GetSalesPersonPerform](@topCount int, @unusedParameter INT)
AS
BEGIN
    DECLARE @unusedVariable INT

    RETURN SELECT TOP(@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
    FROM Sales.SalesOrderHeader
    GROUP BY SalesPersonID
    ORDER BY TotalSales DESC
END
```

Script after refactoring

```
CREATE PROCEDURE [dbo].[GetSalesPersonPerform](@topCount int, @unusedParameter INT)
AS
BEGIN
    DECLARE @unusedVariable INT

    RETURN SELECT TOP(@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
    FROM Sales.SalesOrderHeader
```



```
GROUP BY SalesPersonID
ORDER BY TotalSales DESC
END
```

Example #18: Introduce calculatable column

Object Script

```
CREATE TABLE Customer
(
    FirstName varchar(40),
    CustomerId int Primary Key
)

CREATE TABLE Account
(
    AccountId int Primary Key,
    CustomerId int FOREIGN KEY REFERENCES Customer(CustomerId),
    Balance money
)
```

Generated script

```
ALTER TABLE Customer
    ADD TotalAccountBalance money;
GO
CREATE FUNCTION CalculateTotalBalance(@CustomerId INT)
RETURNS money
AS
BEGIN
    return 0
END
GO
UPDATE Customer
SET
    TotalAccountBalance = dbo.CalculateTotalBalance(CustomerId)
GO
CREATE TRIGGER UpdateCustomerTotalAccountBalance
ON Account
FOR INSERT, DELETE, UPDATE
AS
BEGIN
    UPDATE Customer
    SET
        TotalAccountBalance = dbo.CalculateTotalBalance(inserted.CustomerId)
END
```

Example #19: Find usage {advanced}

Table1 object usage search.

Script before refactoring

```
CREATE TABLE Table1
(
    id int
)

CREATE INDEX Index1 on Table1 (Id)

SELECT * FROM Table1;
```

Script after refactoring

```
CREATE TABLE Table1
(
```



```
    id int
)
CREATE INDEX Index1 on Table1 (Id)
SELECT * FROM Table1;
```

Example #20: Split column

Name column split into three columns - FirstName, MiddleName, LastName.

Object Script

```
CREATE TABLE Customer
(
  Id int,
  Name varchar(100)
);
GO
CREATE FUNCTION GetCustomerNames()
RETURNS TABLE
AS
RETURN SELECT Name FROM Customer
```

Generated script

```
ALTER TABLE Customer ADD FirstName varchar(40);
ALTER TABLE Customer ADD MiddleName varchar(40);
ALTER TABLE Customer ADD LastName varchar(40);
GO
GO
CREATE FUNCTION GetFirstName(@name varchar(100))
RETURNS VARCHAR(40)
AS
BEGIN
    --Need implemented
    RETURN @name
END
GO
CREATE FUNCTION GetMiddleName(@name varchar(100))
RETURNS VARCHAR(40)
AS
BEGIN
    --Need implemented
    RETURN @name
END
GO
CREATE FUNCTION GetLastName(@name varchar(100))
RETURNS VARCHAR(40)
AS
BEGIN
    --Need implemented
    RETURN @name
END
GO
UPDATE Customer
SET
    FirstName = GetFirstName(Name),
    MiddleName = GetMiddleName(Name),
    LastName = GetLastName(Name);
ALTER TABLE Customer DROP COLUMN Name;
GO
ALTER FUNCTION GetCustomerNames()
RETURNS TABLE
AS
```



```
--Need replace column Name on new columns FirstName, MiddleName, LastName
RETURN SELECT Name FROM Customer
GO
```

Example #21: Merge Tables

Object Script

```
CREATE TABLE [dbo].[ErrorLog] (
    [ErrorLogID] [int] IDENTITY(1,1) NOT NULL,
    [ErrorTime] [datetime] NOT NULL CONSTRAINT [DF_ErrorLog_ErrorTime] DEFAULT (getdate()),
    [UserName] [sysname] COLLATE Latin1_General_CS_AS NOT NULL,
    [ErrorNumber] [int] NOT NULL,
CONSTRAINT [PK_ErrorLog_ErrorLogID] PRIMARY KEY CLUSTERED
(
    [ErrorLogID] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]

CREATE TABLE [dbo].[ErrorLog2]
(
    [ErrorState] [int] NULL,
    [ErrorProcedure] [nvarchar] (126) COLLATE Latin1_General_CS_AS NULL,
    [ErrorLine] [int] NULL,
    [ErrorMessage] [nvarchar] (4000) COLLATE Latin1_General_CS_AS NOT NULL,
    [ErrorSeverity] [int] NULL,
    [ErrorLogID] [int] NOT NULL IDENTITY(1, 1)
) ON [PRIMARY]

CREATE PROCEDURE dbo.uspGetLogError
    @ErrorTime [DateTime]
AS
BEGIN
    SET NOCOUNT ON;

    SELECT ErrorLog.ErrorTime, ErrorLog.UserName, ErrorLog2.ErrorMessage
    FROM ErrorLog
        INNER JOIN ErrorLog2
            ON ErrorLog.ErrorLogID=ErrorLog2.ErrorLogID
    WHERE ErrorTime>=@ErrorTime
END;
```

Generated script

```
ALTER TABLE [dbo].[ErrorLog] ADD
    [ErrorState] [int] NULL,
    [ErrorProcedure] [nvarchar] (126) COLLATE Latin1_General_CS_AS NULL,
    [ErrorLine] [int] NULL,
    [ErrorMessage] [nvarchar] (4000) COLLATE Latin1_General_CS_AS NOT NULL,
    [ErrorSeverity] [int] NULL;

UPDATE [dbo].[ErrorLog]
SET
    [ErrorLog].[ErrorState] = [ErrorLog2].[ErrorState],
    [ErrorLog].[ErrorProcedure] = [ErrorLog2].[ErrorProcedure],
    [ErrorLog].[ErrorLine] = [ErrorLog2].[ErrorLine],
    [ErrorLog].[ErrorMessage] = [ErrorLog2].[ErrorMessage],
    [ErrorLog].[ErrorSeverity] = [ErrorLog2].[ErrorSeverity]
FROM [dbo].[ErrorLog2]
WHERE [ErrorLog2].ErrorLogID=[ErrorLog].ErrorLogID

DROP TABLE [dbo].[ErrorLog2];

CREATE PROCEDURE dbo.uspGetLogError
    @ErrorTime [DateTime]
```



```
AS
BEGIN
    SET NOCOUNT ON;

    SELECT ErrorLog.ErrorTime, ErrorLog.UserName, ErrorLog.ErrorMessage
    FROM ErrorLog
    WHERE ErrorTime>=@ErrorTime
END;
```

Example #22: Split Table

[ErrorState], [ErrorProcedure], [ErrorLine], [ErrorMessage], [ErrorSeverity] columns are extracted from V3 ErrorLog table into ErrorLog2 table. Tables linking is executed by ErrorLogID column.

Object Script

```
CREATE TABLE [dbo].[ErrorLog] (
    [ErrorLogID] [int] IDENTITY(1,1) NOT NULL,
    [ErrorTime] [datetime] NOT NULL CONSTRAINT [DF_ErrorLog_ErrorTime] DEFAULT (getdate()),
    [UserName] [sysname] COLLATE Latin1_General_CS_AS NOT NULL,
    [ErrorNumber] [int] NOT NULL,
    [ErrorSeverity] [int] NULL,
    [ErrorState] [int] NULL,
    [ErrorProcedure] [nvarchar] (126) COLLATE Latin1_General_CS_AS NULL,
    [ErrorLine] [int] NULL,
    [ErrorMessage] [nvarchar] (4000) COLLATE Latin1_General_CS_AS NOT NULL,
    CONSTRAINT [PK_ErrorLog_ErrorLogID] PRIMARY KEY CLUSTERED
(
    [ErrorLogID] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]

CREATE PROCEDURE dbo.uspGetLogError
    @ErrorTime [DateTime]
AS
BEGIN
    SET NOCOUNT ON;

    SELECT ErrorLog.ErrorTime, ErrorLog.UserName, ErrorLog.ErrorMessage FROM ErrorLog
    WHERE ErrorTime>=@ErrorTime
END;
```

Generated script

```
CREATE TABLE [dbo].[ErrorLog2]
(
    [ErrorState] [int] NULL,
    [ErrorProcedure] [nvarchar] (126) COLLATE Latin1_General_CS_AS NULL,
    [ErrorLine] [int] NULL,
    [ErrorMessage] [nvarchar] (4000) COLLATE Latin1_General_CS_AS NOT NULL,
    [ErrorSeverity] [int] NULL,
    [ErrorLogID] [int] NOT NULL IDENTITY(1, 1)
) ON [PRIMARY]

INSERT INTO [dbo].[ErrorLog2] ([ErrorState], [ErrorProcedure], [ErrorLine], [ErrorMessage], [ErrorSeverity],
[ErrorLogID])
SELECT DISTINCT [ErrorState], [ErrorProcedure], [ErrorLine], [ErrorMessage], [ErrorSeverity], [ErrorLogID]
FROM [dbo].[ErrorLog]

ALTER TABLE [dbo].[ErrorLog2]
ADD CONSTRAINT [pk_ErrorLog2]
PRIMARY KEY CLUSTERED ([ErrorLogID]) ON [PRIMARY]
```



```
ALTER TABLE [dbo].[ErrorLog] DROP
COLUMN [ErrorSeverity],
COLUMN [ErrorState],
COLUMN [ErrorProcedure],
COLUMN [ErrorLine],
COLUMN [ErrorMessage]

ALTER PROCEDURE dbo.uspGetLogError
    @ErrorTime [DateTime]
AS
BEGIN
    SET NOCOUNT ON;

    SELECT ErrorLog.ErrorTime, ErrorLog.UserName, ErrorLog2.ErrorMessage
    FROM ErrorLog
        INNER JOIN ErrorLog2
        ON ErrorLog.ErrorLogID=ErrorLog2.ErrorLogID
    WHERE ErrorTime>=@ErrorTime
END;
```

Example #23: Moving of column

Object Script

```
CREATE TABLE Customer
(
    FirstName varchar(40),
    CustomerId int Primary Key,
    Balance money
)

CREATE TABLE Account
(
    AccountId int Primary Key,
    CustomerId int FOREIGN KEY REFERENCES Customer(CustomerId)
)
```

Generated script

```
ALTER TABLE Account
    ADD Balance money;

UPDATE Account
    SET Balance = (SELECT Balance FROM Customer
        WHERE CustomerId = Account.CustomerId);

ALTER TABLE Customer
    DROP COLUMN Balance
```

Example #24: Change signature

Order of GetSalesPersonPerform function parameters is changed.

Object Script

```
CREATE Function [dbo].[GetSalesPersonPerform](@topCount int, @minSum money)
RETURNS TABLE
AS
RETURN SELECT TOP(@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
    FROM Sales.SalesOrderHeader
    GROUP BY SalesPersonID
    HAVING Sum(TotalDue)>0
    ORDER BY TotalSales DESC

CREATE PROCEDURE Procedure1
```



```
AS
SELECT * FROM [dbo].[GetSalesPersonPerform] (7, 0)
```

Generated script

```
ALTER Function [dbo].[GetSalesPersonPerform] ( @minSum money, @topCount int)
RETURNS TABLE
AS
RETURN SELECT TOP (@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
HAVING Sum(TotalDue)>0
ORDER BY TotalSales DESC

ALTER PROCEDURE Procedure1
AS
SELECT * FROM [dbo].[GetSalesPersonPerform] (0, 7)
```

Example #25: "Move Schema" Refactoring

ErrorLog table transfer into HumanResources schema.

Object Script

```
CREATE TABLE [dbo].[ErrorLog] (
    [ErrorLogID] [int] IDENTITY(1,1) NOT NULL,
    [ErrorTime] [datetime] NOT NULL CONSTRAINT [DF_ErrorLog_ErrorTime] DEFAULT (getdate()),
    [UserName] [sysname] COLLATE Latin1_General_CS_AS NOT NULL,
    [ErrorNumber] [int] NOT NULL,
    [ErrorSeverity] [int] NULL,
    [ErrorState] [int] NULL,
    [ErrorProcedure] [nvarchar] (126) COLLATE Latin1_General_CS_AS NULL,
    [ErrorLine] [int] NULL,
    [ErrorMessage] [nvarchar] (4000) COLLATE Latin1_General_CS_AS NOT NULL,
    CONSTRAINT [PK_ErrorLog_ErrorLogID] PRIMARY KEY CLUSTERED
)
    ([ErrorLogID] ASC
) WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]

-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
CREATE PROCEDURE [dbo].[uspLogError]
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
                                -- by uspLogError in the ErrorLog table
AS
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;

        -- Return if inside an uncommittable transaction.
        -- Data insertion/modification is not allowed when
        -- a transaction is in an uncommittable state.
        IF XACT_STATE() = -1
            BEGIN
                PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
```



```
information.';
    + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
    RETURN;
END

INSERT [dbo].[ErrorLog]
(
    [UserName],
    [ErrorNumber],
    [ErrorSeverity],
    [ErrorState],
    [ErrorProcedure],
    [ErrorLine],
    [ErrorMessage]
)
VALUES
(
    CONVERT(sysname, CURRENT_USER),
    ERROR_NUMBER(),
    ERROR_SEVERITY(),
    ERROR_STATE(),
    ERROR_PROCEDURE(),
    ERROR_LINE(),
    ERROR_MESSAGE()
);

-- Pass back the ErrorLogID of the row inserted
SET @ErrorLogID = @@IDENTITY;
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;
```

Generated script

```
ALTER SCHEMA HumanResources TRANSFER [dbo].[ErrorLog];

-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
ALTER PROCEDURE [dbo].[uspLogError]
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS
    -- by uspLogError in the ErrorLog table
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;

        -- Return if inside an uncommittable transaction.
        -- Data insertion/modification is not allowed when
        -- a transaction is in an uncommittable state.
        IF XACT_STATE() = -1
            BEGIN
                PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
                + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
                RETURN;
            END
        END TRY
    END CATCH
END;
```



```
END

INSERT [HumanResources].[ErrorLog]
(
    [UserName],
    [ErrorNumber],
    [ErrorSeverity],
    [ErrorState],
    [ErrorProcedure],
    [ErrorLine],
    [ErrorMessage]
)
VALUES
(
    CONVERT(sysname, CURRENT_USER),
    ERROR_NUMBER(),
    ERROR_SEVERITY(),
    ERROR_STATE(),
    ERROR_PROCEDURE(),
    ERROR_LINE(),
    ERROR_MESSAGE()
);

-- Pass back the ErrorLogID of the row inserted
SET @ErrorLogID = @@IDENTITY;
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;
```

Example #26: Replacing auxiliary key with natural key

State table surrogate key is replaced with StateCode natural key.

Object Script

```
CREATE TABLE State
(
    StatePOID int PRIMARY KEY,
    StateCode varchar(4),
    Name varchar(40)
)

CREATE TABLE Address
(
    Street varchar(50),
    State varchar(50),
    StatePOID int FOREIGN KEY REFERENCES,
    PostCode varchar(10),
    CountryCode varchar(2)
)
```

Generated script

```
ALTER TABLE Address ADD StateCode varchar(4) NOT NULL FOREIGN KEY REFERENCES State(StateCode);

ALTER TABLE Address DROP CONSTRAINT AddressToStateForeignKey;
ALTER TABLE State DROP CONSTRAINT StatePrimaryKey;
ALTER TABLE State ALTER COLUMN StateCode varchar(4) NOT NULL;
ALTER TABLE State ADD CONSTRAINT StatePrimaryKey PRIMARY KEY (StateCode);
```

Example #27: Merge columns



FirstName, MiddleName, LastName columns merge into Name column.

Object Script

```
CREATE TABLE Customer
(
  Id int,
  FirtsName varchar(40),
  MiddleName varchar(40),
  LastName varchar(40),
);

GO
CREATE FUNCTION GetCustomerNames()
RETURNS TABLE
AS
RETURN SELECT FirstName, LastName FROM Customer
GO
ALTER TABLE Customer ADD Name varchar(120);
```

Generated script

```
CREATE FUNCTION GetName(@firstName varchar(40), @middleName varchar(40), @lastName varchar(40))
RETURNS VARCHAR(120)
AS
BEGIN
    --Need implemented
    RETURN ''
END
GO

UPDATE Customer
SET
    Name = GetName(FirstName, MiddleName, LastName);

GO
ALTER FUNCTION GetCustomerNames()
RETURNS TABLE
AS
--Need replace columns FirstName, LastName on new column Name
RETURN SELECT FirstName, LastName FROM Customer
GO

ALTER TABLE Customer DROP COLUMN FirstName;
ALTER TABLE Customer DROP COLUMN MiddleName;
ALTER TABLE Customer DROP COLUMN LastName;
```

Example #28: Parameterization of method

GetCustomerByCountry procedure generation from GetAmericanCustomers, GetCanadianCustomers, GetBrazilianCustomers procedures.

Object Script

```
CREATE PROCEDURE GetAmericanCustomers
AS
RETURN SELECT * FROM Customers WHERE Country='US'
GO
CREATE PROCEDURE GetCanadianCustomers
AS
RETURN SELECT * FROM Customers WHERE Country='CA'
GO
CREATE PROCEDURE GetBrazilianCustomers
AS
RETURN SELECT * FROM Customers WHERE Country='BR'
GO
CREATE PROCEDURE Procedure1
AS
```



```
RETURN SELECT * FROM dbo.GetCanadianCustomers()
```

Generated script

```
CREATE PROCEDURE GetCustomerByCountry(@Country varchar(2))
AS
BEGIN
    IF @Country='US'
        RETURN SELECT * FROM Customers WHERE Country='US';

    IF @Country='CA'
        RETURN SELECT * FROM Customers WHERE Country='CA';

    IF @Country='BR'
        RETURN SELECT * FROM Customers WHERE Country='BR';

    RETURN
END
GO
ALTER PROCEDURE Procedure1
AS
RETURN SELECT * FROM dbo.GetCustomerByCountry('CA')
GO
DROP PROCEDURE GetAmericanCustomers;
DROP PROCEDURE GetCanadianCustomers;
DROP PROCEDURE GetBrazilianCustomers;
```

Example #29: Introduce default value

Object script

```
CREATE TABLE Customer
(
    CustomerId int PRIMARY KEY,
    FirstName varchar(40),
    Status varchar(10)
)
```

Generated script

```
CREATE DEFAULT CustomerStatusDefault as 'NEW';
GO
exec sp_bindefault 'CustomerStatusDefault', 'Customer.Status'
```

Example #30: Inline of Stored Procedure

uspPrintError procedure removal.

Object Script

```
-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
CREATE PROCEDURE dbo.uspLogError
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS -- by uspLogError in the ErrorLog table
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
```



```
-- Return if there is no error information to log
IF ERROR_NUMBER() IS NULL
    RETURN;

-- Return if inside an uncommittable transaction.
-- Data insertion/modification is not allowed when
-- a transaction is in an uncommittable state.
IF XACT_STATE() = -1
BEGIN
    PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
        + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
    RETURN;
END

INSERT dbo.NewErrorLog
(
    [UserName],
    [ErrorNumber],
    [ErrorSeverity],
    [ErrorState],
    [ErrorProcedure],
    [ErrorLine],
    [ErrorMessage]
)
VALUES
(
    CONVERT(sysname, CURRENT_USER),
    ERROR_NUMBER(),
    ERROR_SEVERITY(),
    ERROR_STATE(),
    ERROR_PROCEDURE(),
    ERROR_LINE(),
    ERROR_MESSAGE()
);

-- Pass back the ErrorLogID of the row inserted
SET @ErrorLogID = @@IDENTITY;
END TRY
BEGIN CATCH
    PRINT 'An error occurred in stored procedure uspLogError: ';
    EXECUTE [dbo].[uspPrintError];
    RETURN -1;
END CATCH
END;

CREATE PROCEDURE [dbo].[uspPrintError]
AS
BEGIN
    SET NOCOUNT ON;

    -- Print error information.
    PRINT 'Error ' + CONVERT(varchar(50), ERROR_NUMBER()) +
        ', Severity ' + CONVERT(varchar(5), ERROR_SEVERITY()) +
        ', State ' + CONVERT(varchar(5), ERROR_STATE()) +
        ', Procedure ' + ISNULL(ERROR_PROCEDURE(), '-') +
        ', Line ' + CONVERT(varchar(5), ERROR_LINE());
    PRINT ERROR_MESSAGE();
END;
```

Generated script

```
-- uspLogError logs error information in the ErrorLog table about the
-- error that caused execution to jump to the CATCH block of a
-- TRY...CATCH construct. This should be executed from within the scope
-- of a CATCH block otherwise it will return without inserting error
-- information.
ALTER PROCEDURE dbo.uspLogError
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
```



```
AS          -- by uspLogError in the ErrorLog table
BEGIN
    SET NOCOUNT ON;

    -- Output parameter value of 0 indicates that error
    -- information was not logged
    SET @ErrorLogID = 0;

    BEGIN TRY
        -- Return if there is no error information to log
        IF ERROR_NUMBER() IS NULL
            RETURN;

        -- Return if inside an uncommittable transaction.
        -- Data insertion/modification is not allowed when
        -- a transaction is in an uncommittable state.
        IF XACT_STATE() = -1
            BEGIN
                PRINT 'Cannot log error since the current transaction is in an uncommittable state. '
                    + 'Rollback the transaction before executing uspLogError in order to successfully log error
information.';
                RETURN;
            END

        INSERT dbo.NewErrorLog
            (
                [UserName],
                [ErrorNumber],
                [ErrorSeverity],
                [ErrorState],
                [ErrorProcedure],
                [ErrorLine],
                [ErrorMessage]
            )
        VALUES
            (
                CONVERT(sysname, CURRENT_USER),
                ERROR_NUMBER(),
                ERROR_SEVERITY(),
                ERROR_STATE(),
                ERROR_PROCEDURE(),
                ERROR_LINE(),
                ERROR_MESSAGE()
            );

        -- Pass back the ErrorLogID of the row inserted
        SET @ErrorLogID = @@IDENTITY;
    END TRY
    BEGIN CATCH
        PRINT 'An error occurred in stored procedure uspLogError: ';
        SET NOCOUNT ON;

        -- Print error information.
        PRINT 'Error ' + CONVERT(varchar(50), ERROR_NUMBER()) +
            ', Severity ' + CONVERT(varchar(5), ERROR_SEVERITY()) +
            ', State ' + CONVERT(varchar(5), ERROR_STATE()) +
            ', Procedure ' + ISNULL(ERROR_PROCEDURE(), '-') +
            ', Line ' + CONVERT(varchar(5), ERROR_LINE());
        PRINT ERROR_MESSAGE();

        RETURN -1;
    END CATCH
END;

DROP PROCEDURE [dbo].[uspPrintError];
```

Example #31: Inline of Scalar Function



Calculate function removal.

Object Script

```
CREATE FUNCTION Calculate (@value int)
returns float
as
begin
    return log(exp(@value%10) + sin(@value))
end

CREATE PROCEDURE Procedure1 as
Return select dbo.Calculate(object_id), * from sys.objects
```

Generated script

```
ALTER PROCEDURE Procedure1 as
Return select log(exp(object_id%10) + sin(object_id)), * from sys.objects

DROP FUNCTION Calculate;
```

Example #32: Inline of Inline Table-Valued Function

Drops GetSalesPersonPerform procedure.

Object Script

```
CREATE Function GetSalesPersonPerform(@topCount int)
RETURNS TABLE
AS
RETURN SELECT TOP(@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
ORDER BY TotalSales DESC
GO
CREATE PROCEDURE Procedure1(@topCount int)
as
SELECT * FROM Sales.SalesPerson s
WHERE s.SalesPersonID in
(SELECT SalesPersonID FROM dbo.GetSalesPersonPerform(@topCount) TopSalesPerson)
GO
```

Generated script

```
ALTER PROCEDURE Procedure1(@topCount int)
as
SELECT * FROM Sales.SalesPerson s
WHERE s.SalesPersonID in
(SELECT SalesPersonID FROM
(SELECT TOP(@topCount) SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
ORDER BY TotalSales DESC) TopSalesPerson
)
GO
DROP Function GetSalesPersonPerform;
```

Example #33: Inline of View

SalesPersonPerform view removal.

Object Script



```
CREATE VIEW SalesPersonPerform
AS
SELECT TOP 10 SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
ORDER BY TotalSales DESC

CREATE PROCEDURE Procedure1
as
SELECT * FROM Sales.SalesPerson s
WHERE s.SalesPersonID in
(SELECT SalesPersonID FROM SalesPersonPerform TopSalesPerson)
GO
```

Generated script

```
ALTER PROCEDURE Procedure1
as
SELECT * FROM Sales.SalesPerson s
WHERE s.SalesPersonID in
(SELECT SalesPersonID FROM
(SELECT TOP 10 SalesPersonID, SUM(TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID
ORDER BY TotalSales DESC) TopSalesPerson
)
GO

DROP VIEW SalesPersonPerform
```

Example #34: Introduce trigger to collect historical data

Object Script

```
CREATE TABLE Customer
(
    CustomerId int PRIMARY KEY,
    FirstName varchar(40),
    Status varchar(10)
)
```

Generated script

```
CREATE TABLE CustomerHistory
(
    CustomerId int PRIMARY KEY,
    FirstName varchar(40),
    Status varchar(10),
    ChangedOn datetime
)

CREATE TRIGGER UpdateCustomerHistory
ON Customer
AFTER INSERT, DELETE, UPDATE
AS
BEGIN
    IF UPDATE(CustomerId) OR UPDATE(FirstName) OR UPDATE(Status)
    BEGIN
        INSERT INTO CustomerHistory
        VALUES(inserted.CustomerId, inserted.FirstName, inserted.Status, GetDate());
    END
    ELSE
    END
END
```

Example #35: Transform column to allow NULL



Object Script

```
CREATE TABLE Customer
(
  CustomerId int PRIMARY KEY,
  FirstName varchar(40),
  Status varchar(10) NOT NULL DEFAULT 'New'
)
```

Generated script

```
ALTER TABLE Customer
  ALTER COLUMN Status varchar(10) NULL

UPDATE Customer
  SET Status = CASE WHEN Status='New' THEN NULL ELSE Status END;
```