SQL Refactor 1.1

SQL Refactor - 1.1

redgate®
ingeniously simple tools

Contents

Using SQL Refactor

SQL Refactor™ enables you to improve the appearance or structure of your SQL code without changing its behavior.

SQL Refactor integrates with Microsoft SQL Server™ 2005 Management Studio.

The following refactorings are provided:

- Encapsulate As New Stored Procedure creates a new stored procedure from selected SQL code, and optionally replaces the selected code with a reference to the stored procedure.
- Expand Wildcards expands subqueries that contain SELECT * or SELECT *table.** so that they list all of the columns.
- Find Unused Variables and Parameters highlights any variables or parameters in a script that are declared or assigned values but not used, and any values that are not used.
- Lay Out SQL reformats your SQL code to make it more readable without changing its behavior.
- Qualify Object Names modifies the script so that all object names are qualified in the format [owner].[object]
- Smart Rename generates a script to rename objects, columns, or parameters in a database without breaking dependencies so that the integrity of the database is maintained.
- Split Table generates a script that copies and moves columns from an existing table to a new table whilst retaining the original data and modifying referencing objects to maintain database integrity. You can use this feature to create tables to enforce domain restrictions using referential integrity.
- Summarize Script displays a summary of the actions that a script will perform in the order in which they will occur.
- Uppercase Keywords (page 32) puts keywords into uppercase.

You can also set up options for these refactorings, and save them as options sets.

For information on what's in this version, see http://www.red-gate.com/support/page?p=SQL Refactor&c=SQL_Refactor/articles/version_1xx_SQLRefactor.htm (http://www.red-gate.com/support/versions/version1xx_sqlrefactor.htm).

Activating SQL Refactor

When you download a Red Gate Software product, you have a 14 day trial period in which you can evaluate the product. When your trial period expires, you are invited to purchase the product. If you need more time to evaluate the product, contact licensing@red-gate.com (mailto:licensing@red-gate.com)

When you purchase the product, you are sent an invoice that contains your serial number. You use the serial number to activate the product. If you cannot find your invoice, you can review your serial numbers at http://www.red-gate.com/myserialnumbers (http://www.red-gate.com/myserialnumbers). Note that if you purchased a bundle (/support/page?c=all_products%5carticles%5cbundle_history.htm) of products, this serial number activates all of the products in the bundle.

When you activate the product, the activation program sends an activation request to the Red Gate activation server, using checksums of attributes from your computer. The checksums that are sent to the activation server do not contain any details that might pose a security risk.

You can activate SQL Refactor by Internet, by email, or by using manual activation.

Activating by Internet

When you activate by Internet, the Red Gate activation server is contacted and returns an activation response and an encrypted key to unlock the software. Note that you should receive the activation response within a few minutes of sending your request.

If you are experiencing problems when you activate by Internet, try manual activation.

Activating by email

When you activate by email, you must copy all of the activation request provided in the product activation dialog box. Send it in a plain text email to activation@red-gate.com (mailto:activation@red-gate.com); if you use a format other than plain text for your email, activation may fail. Alternatively, you can copy the activation request, save it to a text file, and attach the file to the email.

Do not send multiple requests for activation in the same email, as activation will fail.

You should receive an email from the Red Gate activation server within 30 minutes. When you receive the email from Red Gate Software, open the attached text file, and copy the text into the empty text box on the next page of the product activation dialog box.

If you are experiencing problems when you activate by email, try manual activation.

Activating using manual activation

Use manual activation if:

- the product is installed on a computer that is not connected to the Internet
- your network uses a proxy server that interrupts contact between the product and the Red Gate activation server
- Internet activation is unsuccessful (see Troubleshooting licensing and activation errors (/support/page?c=all_products%5carticles%5clicensing_error_messages.htm))

To use manual activation, select **I would like to activate the product by e-mail**. Copy all of the activation request provided on the product activation dialog box and go to the Manual Activation page (http://www.red-gate.com/activate) of the Red Gate Web site. Paste your activation request into the box under **Step 1** and click **Get Activation Response**. When you receive the activation response under **Step 2**, copy it. On the next page of the product activation dialog box, paste the response.

Encapsulate as new stored procedure

The **Encapsulate As New Stored Procedure** refactoring generates a script to create a new stored procedure from selected SQL code. If required, the originating script is modified to replace the encapsulated code with a reference to the new stored procedure.

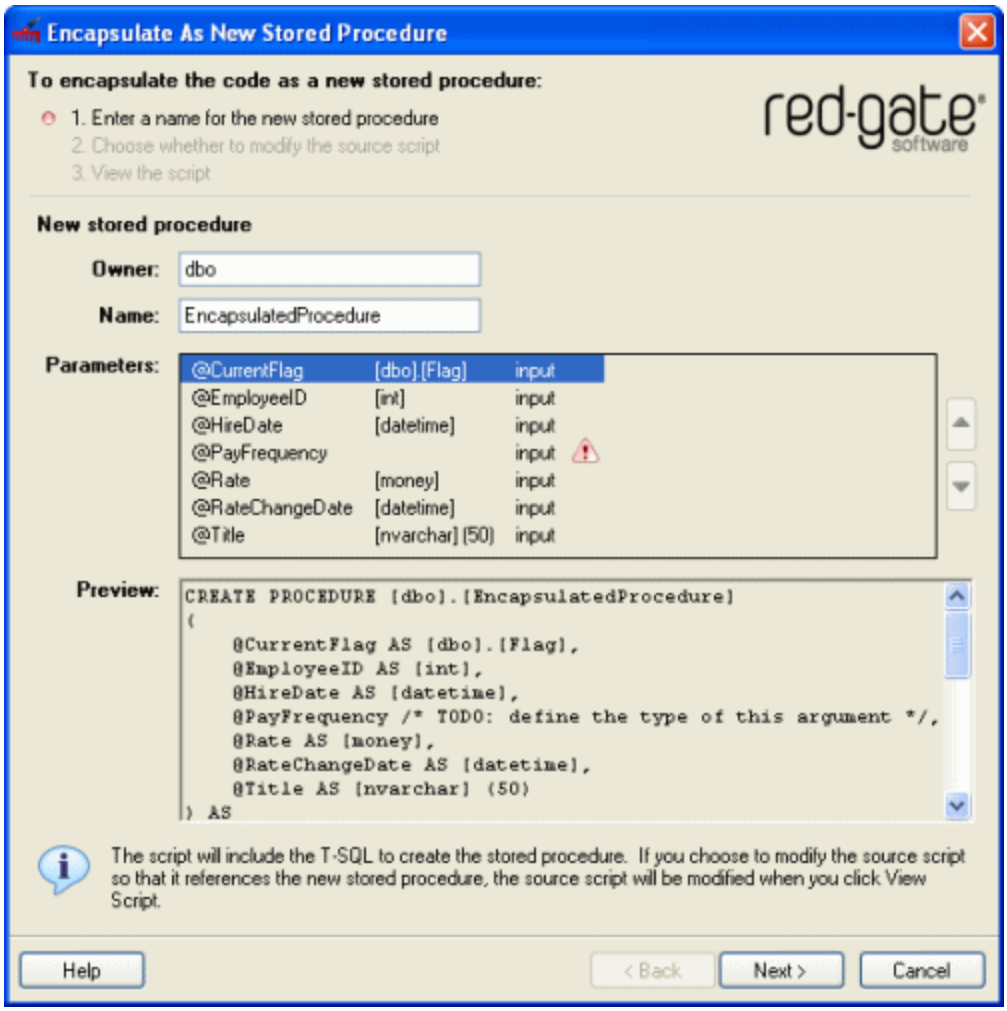
Parameters are created from any variables found in the selected code, therefore you should ensure that you declare any variables within the SQL code that you are encapsulating.

To encapsulate code as a new stored procedure:

1. Open the source script in a SQL Server Management Studio query editor.
2. Select the section of the script that you want to use for the new stored procedure.

   Note that if there are syntax errors in the selected code, SQL Refactor cannot generate the script to create the stored procedure, and an error message will be displayed when you select the feature.
3. On the **SQL Refactor** menu, click **Encapsulate As New Stored Procedure**.

The **Encapsulate As New Stored Procedure** dialog box is displayed.



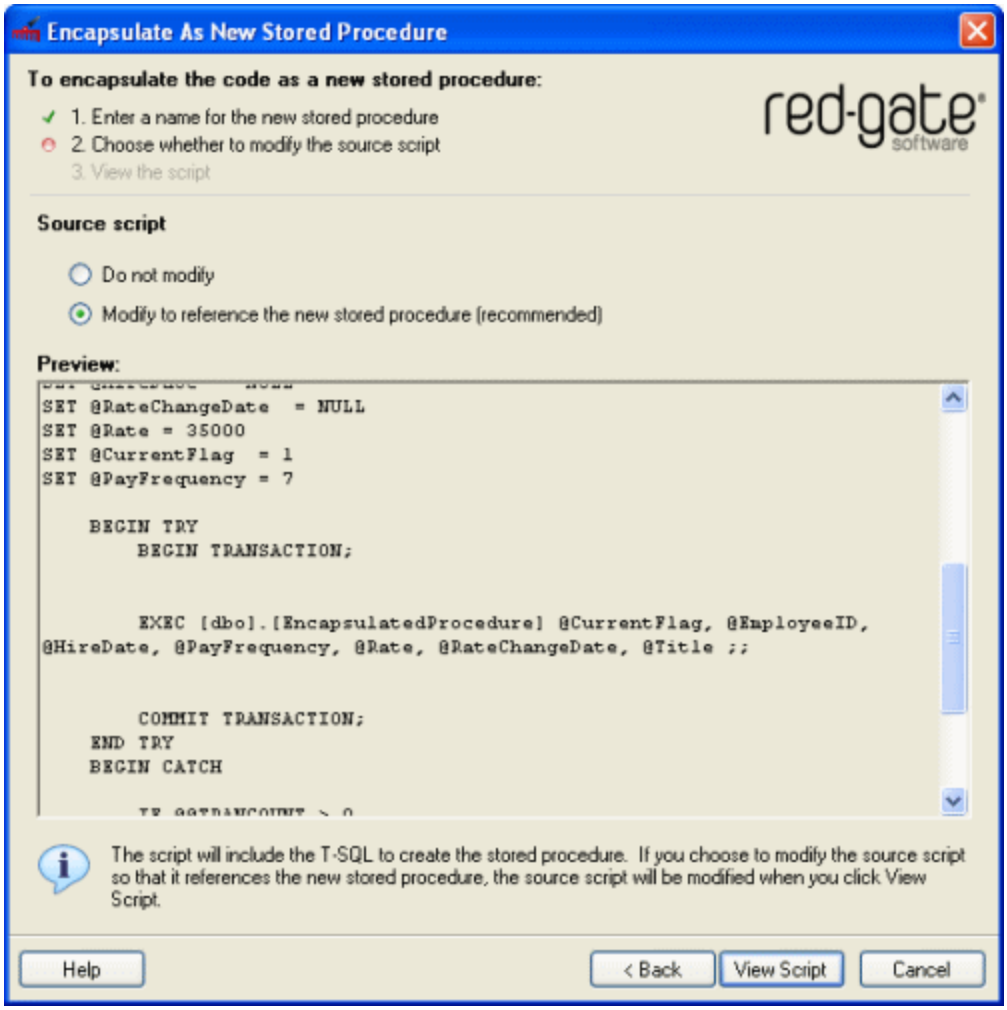
Any variables that will be used to create parameters are listed in **Parameters**. SQL Refactor automatically determines whether each variable will be an input parameter or an output parameter. If a warning triangle is displayed next to a parameter, move your mouse pointer to the warning triangle to see details of the problem.

The SQL code to create the stored procedure is displayed in **Preview**.

4. In the **Owner** box, type the name of the owner for the new stored procedure.
5. In the **Name** box, type a name for the new stored procedure.

   Note that SQL Refactor does not check for duplicate names. If a stored procedure with this owner and name already exists, the script will fail when you run it.
6. Reorder the **Parameters** as required using the and buttons.
7. Click **Next**.

SQL Refactor displays the *source* code in the **Preview** box, and shows how it will be modified to replace your selected code with a reference to the new stored procedure. If required, you can change the **Source script** option so that the source code is not modified.



8. Click **View Script**.

   The **Encapsulate As New Stored Procedure** dialog box is closed, and the SQL script to create the stored procedure is displayed in a new query editor.

   In the source code, SQL Refactor removes the selected code and replaces it with a reference to the new stored procedure (unless you chose not to do this).

9. If necessary, edit the generated stored procedure creation script, and run the script to create the new stored procedure.
10. If you are happy with the changes to the source code, save the changes.

Expand wildcards

You can use **Expand Wildcards** to expand SELECT * and SELECT *table*.* statements so that they list all of the columns that exist in the referenced tables.

SQL Refactor takes USE *database* statements into account to ensure that the correct columns are identified.

For example:

```
SELECT
    *
FROM [HumanResources].[Employee] e
    INNER JOIN [Person].[Contact] c
    ON c.[ContactID] = e.[ContactID]
    INNER JOIN [HumanResources].[EmployeeDepartmentHistory] edh
    ON e.[EmployeeID] = edh.[EmployeeID]
    INNER JOIN [HumanResources].[Department] d
    ON edh.[DepartmentID] = d.[DepartmentID]
WHERE GETDATE() BETWEEN edh.[StartDate] AND ISNULL(edh.[EndDate], GETDATE());
```

Would become:

```
SELECT
    [e].[EmployeeID], [e].[NationalIDNumber], [e].[ContactID], [e].[LoginID], [e].[ManagerID], [e].
[Title], [e].[BirthDate], [e].[MaritalStatus], [e].[Gender], [e].[HireDate], [e].[SalariedFlag], [e].
[VacationHours], [e].[SickLeaveHours], [e].[CurrentFlag], [e].[rowguid], [e].[ModifiedDate], [c].
[ContactID], [c].[NameStyle], [c].[Title], [c].[FirstName], [c].[MiddleName], [c].[LastName], [c].
[Suffix], [c].[EmailAddress], [c].[EmailPromotion], [c].[Phone], [c].[PasswordHash], [c].
[PasswordSalt], [c].[AdditionalContactInfo], [c].[rowguid], [c].[ModifiedDate], [edh].[EmployeeID],
[edh].[DepartmentID], [edh].[ShiftID], [edh].[StartDate], [edh].[EndDate], [edh].[ModifiedDate], [d].
[DepartmentID], [d].[Name], [d].[GroupName], [d].[ModifiedDate]
FROM [HumanResources].[Employee] e
    INNER JOIN [Person].[Contact] c
    ON c.[ContactID] = e.[ContactID]
    INNER JOIN [HumanResources].[EmployeeDepartmentHistory] edh
    ON e.[EmployeeID] = edh.[EmployeeID]
    INNER JOIN [HumanResources].[Department] d
    ON edh.[DepartmentID] = d.[DepartmentID]
WHERE GETDATE() BETWEEN edh.[StartDate] AND ISNULL(edh.[EndDate], GETDATE())
```

To do this:

1. Open the source script in a SQL Server Management Studio query editor.
2. Ensure you are connected to the SQL Server and database that contains the objects referenced in the SELECT statements.

   Note that if the code contains USE *database* statements, SQL Refactor will change the connection as appropriate.
3. If required, select the code that contains the SELECT statements that you want to expand.

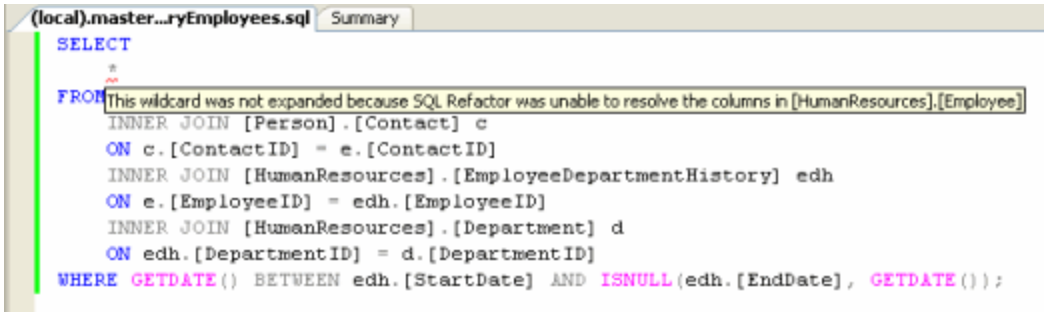
   If you do not select any code, all SELECT statements in the script are expanded.
4. On the **SQL Refactor** menu, click **Expand Wildcards**.

A progress dialog box is displayed. SQL Refactor connects to the SQL Server to expand all SELECT * and SELECT *table.** statements in the code that you selected.

You can undo the changes to the script using the standard SQL Server Management Studio Undo features.

To keep the changes, save the script in the usual way.

If any wildcard characters could not be expanded, for example because the referenced table does not exist, an error dialog box is displayed and the wildcard characters are underlined in red in the query editor. To see error details for an individual wildcard character, move your mouse pointer to the character in the query editor. For example:



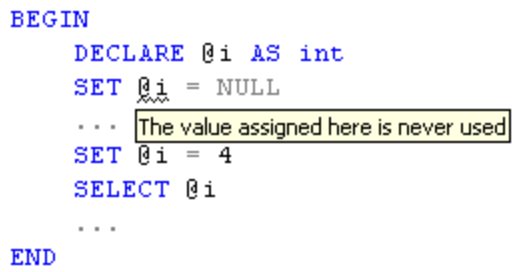
See also

Find unused variables and parameters

SQL Refactor can highlight any parameters or variables that are unused in the current script.

A parameter or variable is considered to be unused if it has been declared or assigned a value, but not queried or used in any control statements, and not used in INSERT or UPDATE statements, WHERE clauses, PRINT or EXECUTE statements, loops or case statements, and so on.

Parameter and variable values that are assigned but not used are also highlighted. In the following example, *@i* is assigned the value *NULL*, but this value is not used before *@i* is assigned the value *4*.



To find unused variables and parameters:

1. Open the source script in a SQL Server Management Studio query editor.
2. If required, select the code that contains the parameters and variables that you want to check.

   If you do not select any code, all parameters and variables in the script are checked.
3. On the **SQL Refactor** menu, click **Find Unused Variables and Parameters**.

   SQL Refactor underlines the unused parameters, variables, and values in the query editor.

See also

Using SQL Refactor 3

Lay out SQL

The **Lay Out SQL** refactoring reformats your SQL code to make it more readable without changing its behavior.

To lay out the SQL code in the current SQL Server Management Studio query editor:

1. Set the SQL Layout options as required.
2. On the **SQL Refactor** menu, click **Lay Out SQL**.

   SQL Refactor lays out your code according to the options you have set. Note that not all commands are fully supported by the **Lay Out SQL** refactoring; for these commands only the wrapping option is applied.

If your SQL code contains syntax errors, SQL Refactor cannot lay out the code. An error is displayed, and the incorrect code is underlined in the query editor.

See also

Qualify object names

The **Qualify Object Names** refactoring modifies the SQL script so that all object names are qualified in the format:

```
[owner].[object]
```

SQL Refactor takes USE *database* statements into account to ensure that the correct owners are identified.

Note that the SQL Server name and database name qualifiers are not added to the names.

To qualify object names:

1. Open the script in a SQL Server Management Studio query editor.
2. Ensure you are connected to the SQL Server and database that contains the objects that are referenced in the script.

   Note that if the selected code contains USE *database* statements, SQL Refactor will change the connection as appropriate.
3. If required, select the code that contains the object names that you want to qualify.

   If you do not select any code, the all object names in the script are qualified.
4. On the **SQL Refactor** menu, click **Qualify Object Names**.

A progress dialog box is displayed. SQL Refactor connects to the SQL Server to find out the owner of each object, and then modifies the script.

You can undo the changes to the script using the standard SQL Server Management Studio Undo features.

To keep the changes, save the script in the usual way.

If any object names could not be qualified, for example because they cannot be found in the database, an error dialog box is displayed and the object names are underlined in red in the query editor.

See also

Using SQL Refactor 3
Script options 35

Smart rename

The **Smart Rename** refactoring generates a script to rename objects in your database without breaking dependencies so that the integrity of your database is maintained. You can also use this feature to change the object owner.

You can rename the following:

- database objects:
  - tables
  - views
  - stored procedures
  - user-defined functions
- parameters in:
  - stored procedures
  - user-defined functions
- columns in:
  - tables
  - views

You can use **Smart Rename** to rename encrypted objects in SQL Server 2000 if you are a member of the *sysadmin* server role; you cannot use this feature to rename encrypted objects in SQL Server 2005.

You are advised to back up your database prior to running the rename script.

When an object is renamed, SQL Refactor:

- changes the definition

  When you run the Smart Rename script, the object definition is modified and SQL Refactor ensures that the *sys.sql_modules* table (for SQL Server 2005) or the *syscomments* table (for SQL Server 2000) remains consistent.
- modifies any objects that reference the renamed object so that dependencies are not broken

  The following referencing objects are modified if necessary:
  - procedures
  - views
  - functions
  - DML triggers
  - DDL triggers
  - queues (if the owner or name of a stored procedure is changed)
  - defaults (if the owner or name of a function is changed)

    - computed columns (if the owner or name of a function is changed)
- changes self-referencing (for recursive stored procedures and functions)
- fixes the name of the object to be renamed, if necessary

  If you have previously renamed an object using SQL Server Management Studio or Enterprise Manager **Rename** or the Transact-SQL **sp_rename** command, the object definition will contain the original name. SQL Refactor fixes the object definition so that it is consistent with the new name. For more information about the **sp_rename** command, see *SQL Server Books Online*.

  Note that any objects that reference the original name (the name prior to using **sp_rename**) are *not* updated.
- fixes the name of any referencing objects that are to be modified, if necessary

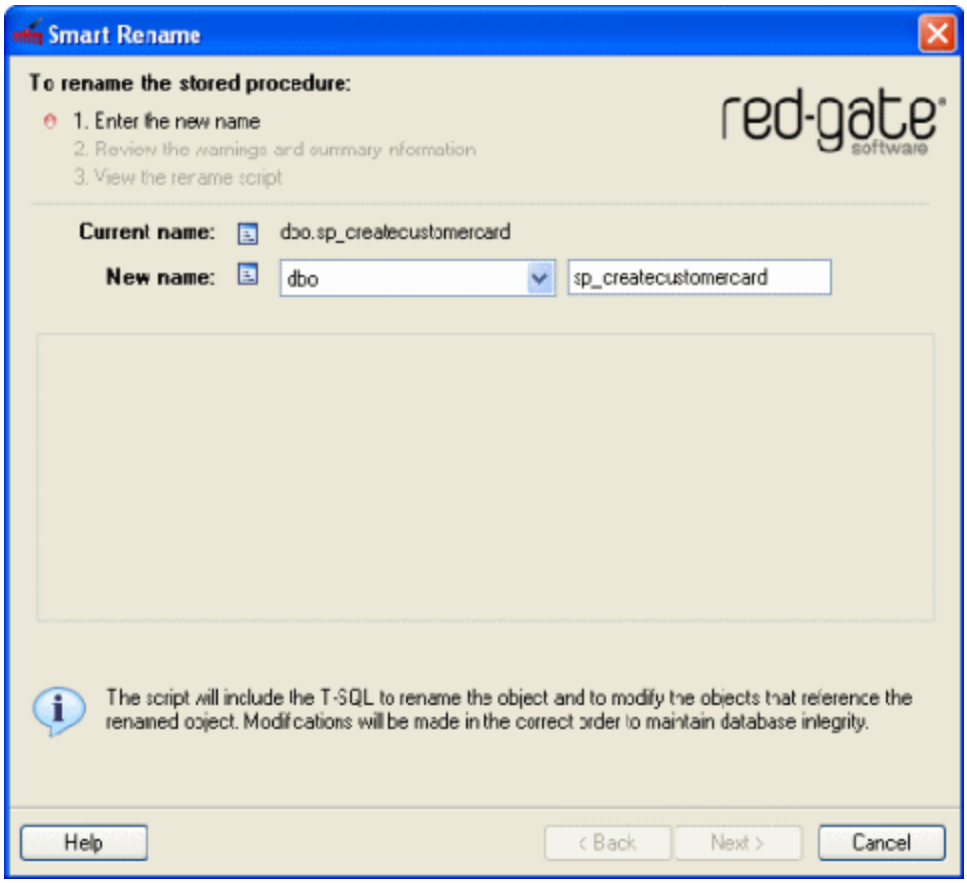
  If you have previously renamed a referencing object using SQL Server Management Studio **Rename** or the Transact-SQL **sp_rename** command, the object definition will contain the original name. SQL Refactor fixes the object definition of the referencing object so that it is consistent with its current name.

The original permissions and extended properties of the object are preserved.

To rename an object:

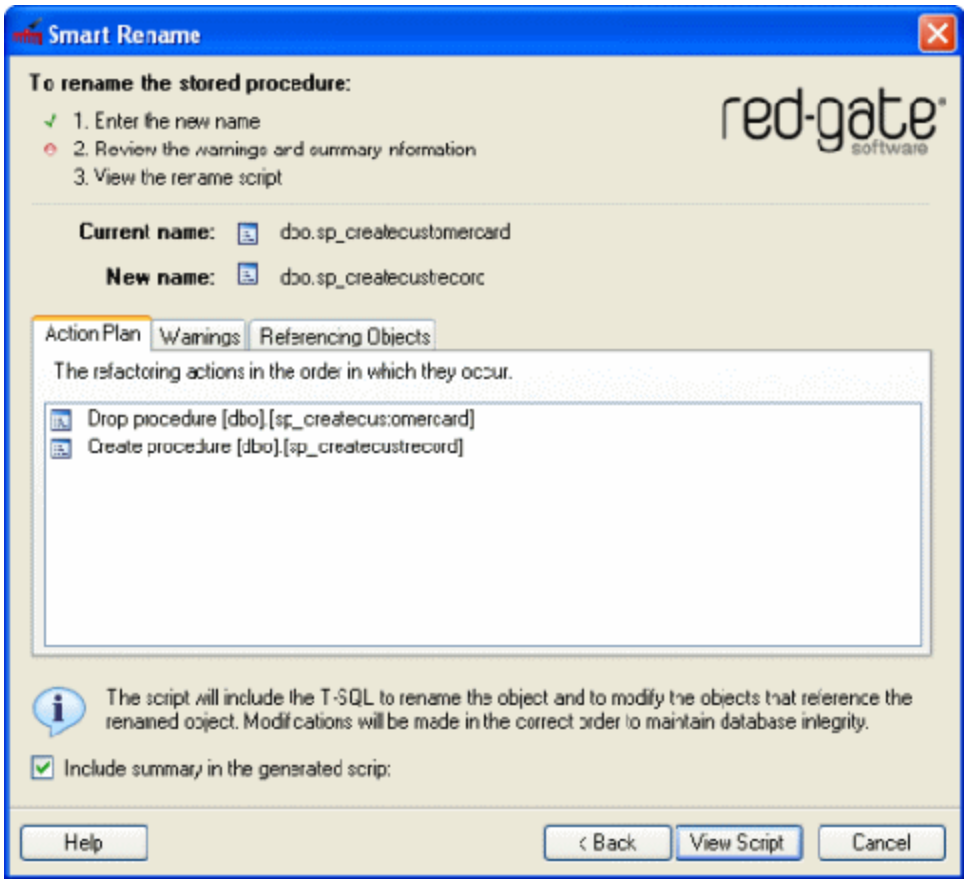
1. On the **SQL Refactor** menu click **Smart Rename**, or in the **Object Explorer** pane, right-click the object and click **Smart Rename**.

The **Smart Rename** dialog box is displayed.



2. If you want to change the object owner, select the name of the new owner.
3. If you want to change the object name, type the new name.
4. Click **Next**.

SQL Refactor checks that the name is not already in use. If the name is valid, SQL Refactor generates the script and displays summary information:



- **Action Plan** is a summary of the actions that the script will perform, in the order in which they will occur.
- **Warnings** displays information that you should consider prior to running the script, and reasons the script might fail.

  The warnings are graded according to severity.
- **Referencing Objects** lists the objects that will be modified by the script because they reference the object that is being renamed.

The action plan and warnings are included in the generated script, below the header. If you do not want to include this information in the script, clear the **Include summary in the generated script** check box.

5. Click **View Script**.

   The **Smart Rename** dialog box is closed, and the SQL script is displayed in the SQL Server Management Studio query editor. Review the script, and then run it to rename the object and modify the referencing objects.

Note that when you run the script, the Object Explorer is not updated with the new name. You must refresh the list to update the name that is displayed.

Split table

You may want to split a table into two separate tables, for example if new requirements arise, or if you need to enforce referential integrity on a set of columns. The **Split Table** refactoring enables you to do this, by generating a script to move or copy columns to a new table whilst retaining the original data in the table. The script also modifies referencing objects so that queries that accessed the original table return the same result after the split has taken place.

To use **Split Table** to create constraints with referential integrity, *copy* the required columns to the new table (but do not *move* any columns). The new table is then the referential integrity table. Referencing objects are not modified.

In this feature, the existing table that you want to split is called the *primary* table; the new table to which the columns are moved or copied is called the *secondary* table.

Before you split a table, you are recommended to read this topic to ensure you understand fully what will happen when you split the table.

- Preparing the table (page 18) describes some points you may want to consider prior to generating the split table script.
- Splitting the table (page 20) describes how you use the **Split Table** feature to generate the script.
- The split table script (page 26) describes in detail what happens to your tables and referencing objects when you run the script.
- Script failure or cancellation (page 29) describes what happens if the script fails or is cancelled.

You are advised to back up your database prior to running the split table script.

Preparing the table

You may wish to consider the following points to plan how the columns are to be split and to prepare your data appropriately. You should also read The split table script (page 26) for a full understanding of what will happen when the table is split.

When you split a table, the primary table retains its name. You supply the owner and name for the secondary table, and choose which columns are to be included in the secondary table.

Note the following:

Shared columns

You must ensure that at least one column is copied to the secondary table so that it exists in both tables (is *shared*). SQL Refactor uses the shared columns to create the foreign key, and the primary key for the secondary table. Therefore, before you select the **Split Table** feature, you must ensure that the primary table contains the columns that you want to be shared. The values contained within the shared columns must uniquely identify each row of data in the secondary table.

Data

For columns that remain in the primary table, all data is retained.

For columns that are copied or moved to the secondary table, SQL Refactor uses the DISTINCT keyword when possible to ensure that only unique data is copied or moved.

However, for the following data types the DISTINCT keyword cannot be used:

- XML data
- SQL Server 2000 large object (LOB) data (*text*, *ntext*, *image*)

Therefore, if you choose to move or copy any of these data types, SQL Refactor cannot use the DISTINCT keyword when the secondary table is populated. This means that all data is moved or copied to the secondary table, including any duplicate data. If the shared columns contain duplicate data, the primary key cannot be created on these columns and the script will fail. Therefore, if you know you will be moving or copying one of the data types listed above, prior to splitting the table you must ensure that there is no duplicate data in any of the columns that will be shared.

Computed columns

If you move or copy a computed column, you must ensure that any columns that are referenced by the computed column are also moved or copied. If you do not do this, SQL Refactor displays a warning, and you cannot generate the script.

If you copy a computed column and all its referenced columns, and the computed column is persisted, SQL Refactor converts the computed column to a normal column in the secondary table so that data is retained. For example, a column that computes two columns that have data type *char(50)* is changed to a normal column of data type *char(100)*.

If you copy a computed column and all its referenced columns, and the computed column is *not* persisted, SQL Refactor displays a warning; if you have chosen to create the foreign key on the secondary table, the script will fail because it is not possible to create a foreign key on a column that is not persisted.

If you move a computed column and all its referenced columns, the computed column remains a computed column in the secondary table.

You cannot move a column from the primary table if it is required by a computed column in the primary table (but you can copy it).

XML columns

XML columns cannot be shared because a primary key cannot be created on an XML column.

The DISTINCT keyword cannot be used with XML columns; for details, see *Data*.

Timestamp columns

Timestamp columns cannot be copied or moved. This is because data cannot be inserted into timestamp columns and the original values cannot, therefore, be inserted into the secondary table.

Common language runtime data

If a common language runtime (CLR) data column is copied or moved to the secondary table and the CLR data is not byte ordered (IsByteOrder is *false*), the script

will fail. This is because the DISTINCT keyword cannot be used on a CLR column that is not byte ordered

Large object data

Columns that contain large object (LOB) data cannot be shared because a primary key cannot be created on LOB data.

For SQL Server 2000 databases, the DISTINCT keyword cannot be used with LOB data; for details, see *Data*.

Null values

Null values are not allowed in shared columns in the secondary table. This is because the shared columns are used to create the primary key on the secondary table, and primary key columns cannot contain null values.

If any columns that you share (*copy* to the secondary table) allow null values, a warning is displayed. When the script is run, these columns will be modified to disallow nulls (set to NOT NULL).

If any of the data in the shared columns contains null values, the script will fail when it is run.

Identities

If you copy or move an identity column, the identity is copied or moved, and data will be inserted using the IDENTITY_INSERT setting.

Partition schemes

You cannot move a column over which the primary table is partitioned to the secondary table. If you attempt to do this, SQL Refactor displays a warning. You can, however, copy the column if required.

DML triggers

All DML triggers that access the data in the primary table are dropped. You are recommended to save these triggers prior to running the split table script if you will want to recreate them following the split.

Triggers that do not access the primary table are preserved, but are not duplicated on the secondary table.

Splitting the table

To split a table:

1. On the **SQL Refactor** menu click **Split Table**, or in the Object Explorer pane, right-click the table and click **Split Table**.

The **Split Table** dialog box is displayed.

2. In **Secondary table name**, type the name for the table to be created.

   If required, you can change the owner for this table by selecting from the list.

   SQL Refactor checks that you have entered a valid name and the name is not already in use.

3. Click **Next**.

4. Copy or move columns from the primary table to the secondary table as required.

   Note that you must *copy* at least one column so that the tables share a column; for more information, see Shared columns. If a column is shared, is displayed next to its name.

   - To copy a column, select the column in the primary table list and click **Copy >**.
   - To move a column, select the column in the primary table list and click **Move >**.

     You cannot move primary key columns from the primary table. However, you can copy these columns if required.
   - To remove a column from the secondary table, select the column and click **Remove**.

     For example, you may want to do this if you have moved or copied a column in error.

Columns that were *moved* to the secondary table are moved back to the primary table; you cannot remove a column from both tables.

Note that if you move a column to the secondary table and subsequently remove it, it is added to the end of the primary table's list of columns. However, when the tables are split, the columns in the primary table will **not** be reordered; they will remain in the original order.

- To remove all of the columns from the secondary table, click **Reset All**.

You can select multiple columns to move, copy, or remove by using Ctrl and Shift in the usual way. However, if any columns in your selection cannot be moved, copied, or removed the relevant buttons are not available. For example, you cannot copy a column that has already been copied. The lower section of the dialog box displays messages or warnings about your column selection.

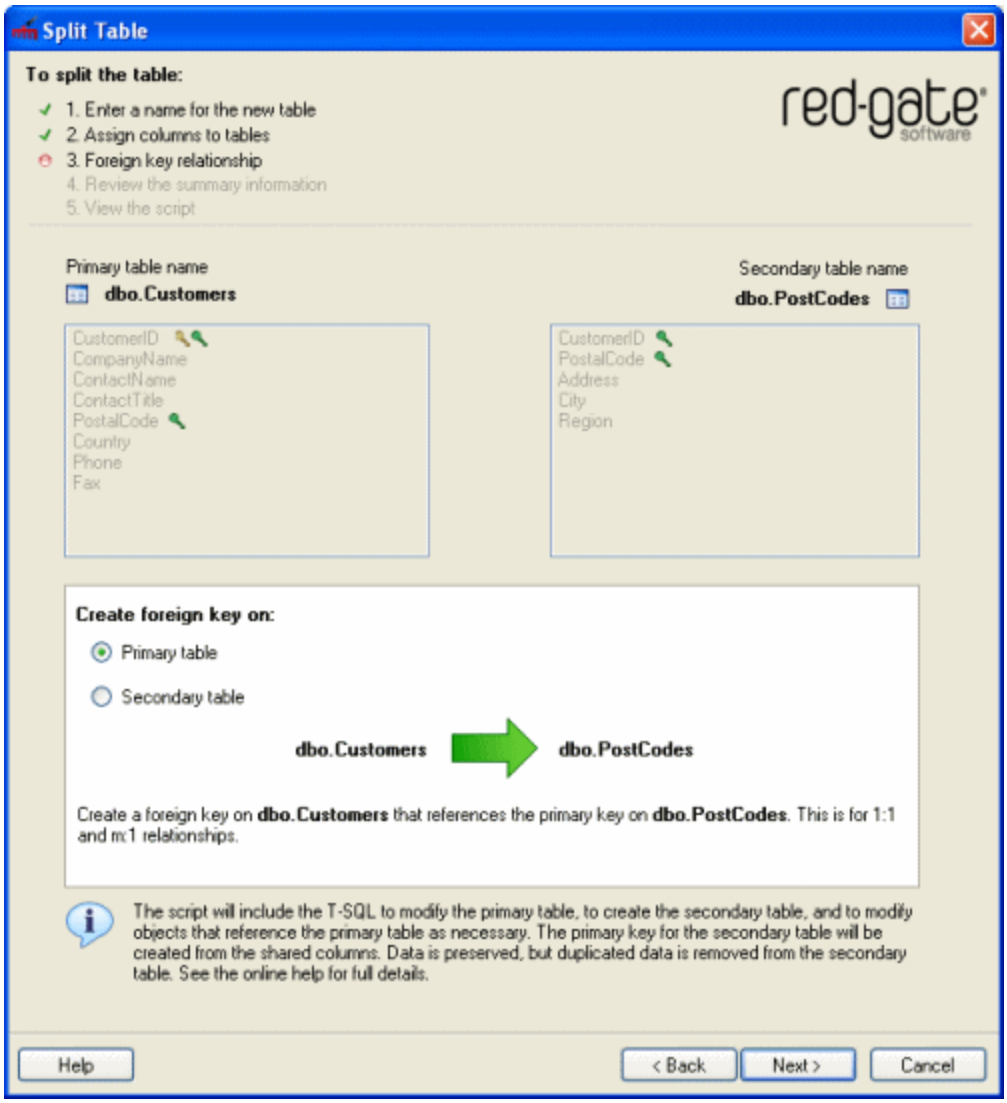
5. Reorder the columns in the secondary table as required using the  and  buttons.

   You can reorder multiple columns at the same time.

   Note that reordering the columns will affect the order of the primary key columns on the secondary table.

6. Click **Next**.

SQL Refactor displays the options for creating the foreign key.

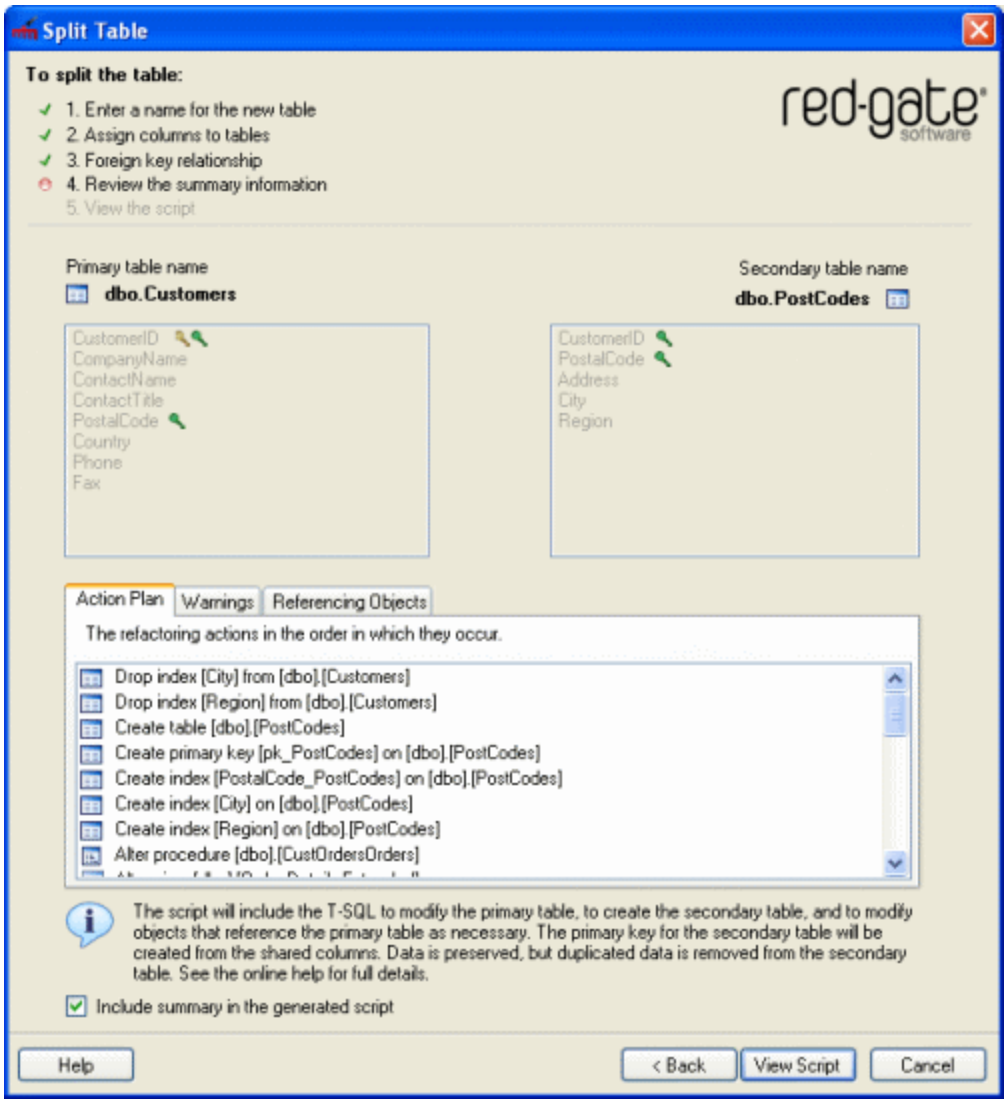


By default, the foreign key is created on the primary table, and it references the primary key on the secondary table. This is for 1:1 or m:1 relationships.

If you have copied all the primary key columns from the primary table to the secondary table, you can choose to create the foreign key on the secondary table, which will reference the primary key on the primary table. This is for 1:1 or 1:n relationships.

7. Click **Next**.

SQL Refactor generates the script and displays summary information:



- **Action Plan** is a summary of the actions that the script will perform in the order in which they will occur.
- **Warnings** displays information that you should consider prior to running the script, and reasons the script may fail (page 29).

  The warnings are graded according to the severity.
- **Referencing Objects** lists the objects that will be modified by the script because they reference the table that is being split.

  The action plan and warnings are included in the generated script, below the header. If you would prefer not to include this information in the script, clear the **Include summary in the generated script** check box.

1. Click **View Script**.

The **Split Table** dialog box is closed, and the SQL script to split the table and modify the referencing objects is displayed in the SQL Server Management Studio query editor.

The Split Table script

When you run the script to split a table the secondary table is created and populated with the data. The primary table, and referencing objects are modified. Details are provided below.

Primary keys

The primary key in the primary table is unchanged.

A primary key for the secondary table is created based on the *shared* columns. SQL Refactor generates a name for the primary key on the secondary table automatically.

The primary key columns cannot contain null values. Therefore, if any of the columns that you choose to share allow null values, a warning is displayed. When the script is run, these columns will be modified to disallow nulls. If any of the data in the shared columns contains null values, the script will fail when it is run.

Note that if you copy the primary key columns from the primary table to the secondary table, the primary key itself is *not* copied; a new primary key is always created on the secondary table. Similarly, any options set on the primary table's primary key (using the WITH clause, such as clustering) are not copied.

Foreign keys

By default, a foreign key is created on the primary table to reference the shared columns in the secondary table (which are used as the secondary table's primary key). SQL Refactor generates a name for the foreign key automatically.

If you chose to create the foreign key on the secondary table, SQL Refactor creates a foreign key on the secondary table to reference the primary key columns in the primary table.

Existing foreign keys that reference other tables:

- on columns in only the primary table, are preserved on the primary table
- on columns in only the secondary table, are created on the secondary table
- on columns that are shared, are preserved on the primary table and duplicated on the secondary table
- on columns in both tables that are not shared, are deleted

Permissions

Table-level permissions on the primary table are duplicated on the secondary table.

Column-level permissions:

- on columns in only the primary table, are preserved on the primary table
- on columns in only the secondary table, are created on the secondary table
- on shared columns, are preserved on the primary table and duplicated on the secondary table

Indexes

Indexes that reference:

- columns in only the primary table, are preserved on the primary table
- columns in only the secondary table, are created on the secondary table
- shared columns, are preserved on the primary table and duplicated on the secondary table
- columns from both tables that are not shared, are dropped

Note that clustered indexes are created as nonclustered indexes on the secondary table. This is because the primary key on the secondary table is clustered to improve access speed when the tables are joined in queries (for more information, see *Referencing objects*).

Constraints

Table-level constraints:

- that reference columns in only the primary table, are preserved on the primary table
- that reference columns in only the secondary table, are created on the secondary table
- that reference shared columns, are preserved on the primary table and duplicated on the secondary table
- that reference columns from both tables that are not shared, are dropped

Column-level constraints:

- on columns in only the primary table, are preserved on the primary table
- on columns in only the secondary table, are created on the secondary table
- on shared columns, are preserved on the primary table and duplicated on the secondary table

Note that DEFAULT constraints are renamed when they are created on the secondary table; the name of the secondary table is appended to the original constraint name. For example, a DEFAULT constraint called *ConstraintA* that is created on a secondary table called *TableB* is created as *ConstraintA_TableB*.

Extended properties

Table-level extended properties are preserved on the primary table and duplicated on the secondary table.

Extended properties on level 2 objects (columns, constraints, triggers, and indexes):

- on objects in only the primary table, are preserved on the primary table
- on objects in only the secondary table, are created on the secondary table
- on shared objects, are preserved on the primary table and duplicated on the secondary table
- on objects from both tables that are not shared, are dropped (because the objects are dropped)

Filegroups If the primary table was created on a filegroup, the secondary table is created on the same filegroup.

Partition schemes

If the primary table has been partitioned over a column, and that column stays in the primary table following the split, the primary table remains partitioned.

If the column over which the table is partitioned is moved to the secondary table, a warning is displayed and SQL Refactor does not allow you to generate the script.

Full-text indexes

If a full-text index exists on a column that is moved to the secondary table, the full-text index is added to the column in the secondary table.

However, note that full-text indexing cannot be added to a column from within a transaction. Therefore, if the script fails, SQL Refactor will not be able to roll back the script, and your database will be in an undetermined state. If you choose to move a column that has a full-text index, SQL Refactor displays a warning.

Referencing objects

If you have only *copied* columns from the primary table to the secondary table, referencing objects are not modified.

If you have *moved* any columns from the primary table to the secondary table, referencing objects are modified as follows.

For the object types listed below, objects that referenced the primary table before it was split are modified so that they reference both tables.

- functions
- stored procedures
- views
- DML triggers
- DDL triggers

The following modifications are made to SELECT statements:

- the table reference is replaced with a JOIN subquery that joins the primary and secondary tables based on the shared columns
- an alias is created for the JOIN subquery
- the SELECT column list and the following clauses are modified to reference the appropriate columns
- WHERE
- GROUP BY
- ORDER BY
- HAVING
- SELECT * clauses are expanded
- wherever possible, fully-qualified names are used
- table hints are preserved for the primary table and duplicated for the secondary table
- table samples are removed

The following modifications are made to INSERT, UPDATE, and DELETE statements:

- INSERT statements are split into two: one for the primary table, and one for the secondary table
- the column list and WHERE clause are modified to reference the appropriate columns
- for statements that include a FROM clause, the table reference is replaced with a JOIN statement that joins the primary and secondary tables based on the shared columns

Note that the generated script for INSERT, UPDATE, and DELETE statements may not precisely reflect your intentions for the data, particularly for complex requirements. Therefore, you should review these statements in detail before you run the script. Foreign keys that reference the primary key on the primary table are not changed.

All DML triggers that accessed columns on the primary table before the split are dropped.

Script failure or cancellation

If a script fails, or if it is cancelled, in most circumstances changes are rolled back. SQL Refactor uses transactions to do this.

However, there are some circumstances in which this is not possible, for example if full-text indexes are to be added to the secondary table. In these cases, SQL Refactor rolls back all the changes that it can. Your database will be in an undetermined state.

SQL Refactor always warns you if it will be unable to roll back changes.

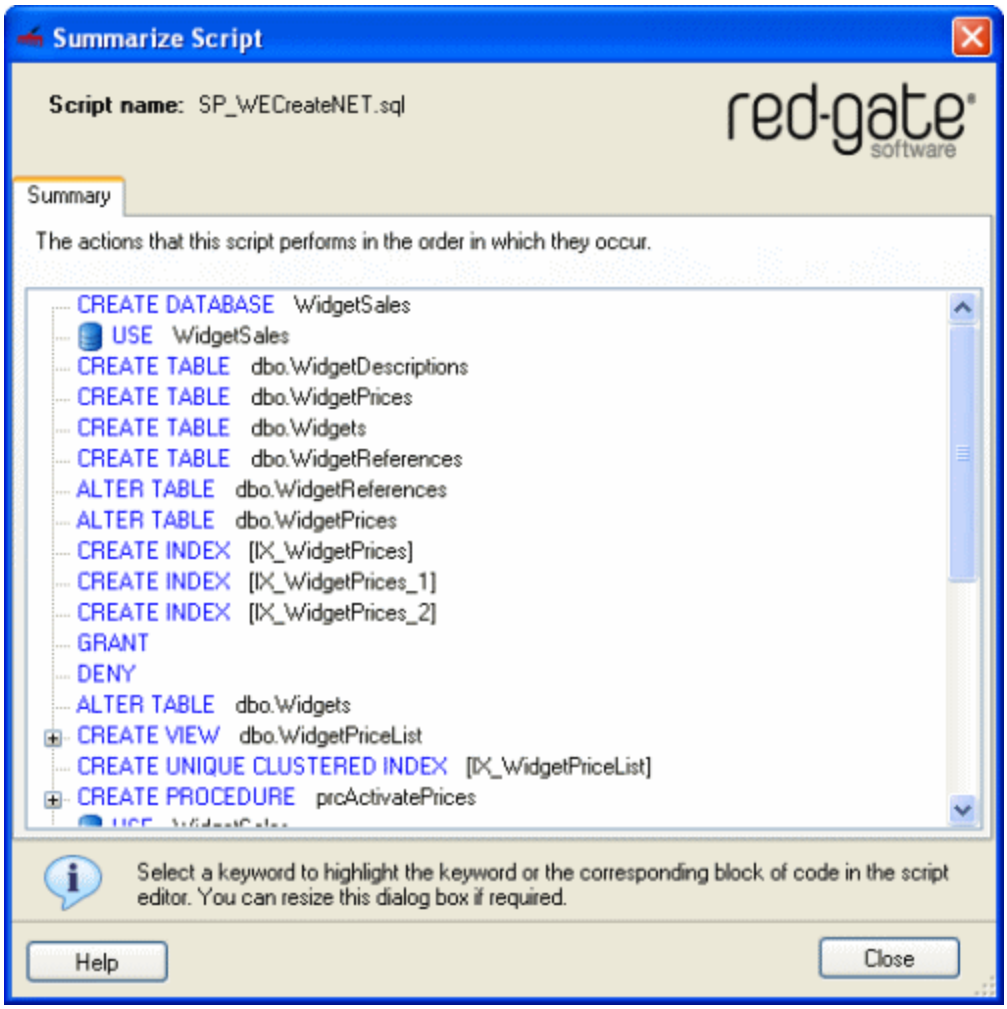
Summarize script

You use **Summarize Script** to view a summary of the actions that the script will perform, in the order in which they will occur.

To see the script summary:

1. Open the script in a SQL Server Management Studio query editor.
2. On the **SQL Refactor** menu, click **Summarize Script**.

   The script summary is displayed in a dialog box:



   SQL Refactor uses a tree structure to group blocks of code. For example, the keywords within a CREATE statement are grouped together. To expand a block, click ⊞ or double-click the keyword.

   When you select a keyword in the summary, it is highlighted in the query editor. If you click on the main keyword for a block, the block is highlighted in the query editor.

The following symbols are used to draw your attention to commands that change the context:

 USE commands

 SETUSER, EXECUTE AS LOGIN, and REVERT commands

You can resize the dialog box to view more of the summary, if required.

3. Click **Close** to close the summary.

Uppercase keywords

You can use **Uppercase Keywords** to put selected keywords in a script into all uppercase.

To do this:

1. Open the source script in a SQL Server Management Studio query editor.
2. Select the code that contains the keywords that you want to change to uppercase.

   If you do not select any code, all keywords in the script are changed to uppercase.
3. On the **SQL Refactor** menu, click **Uppercase Keywords**.

   All keywords that are contained within the code that you selected are expanded.

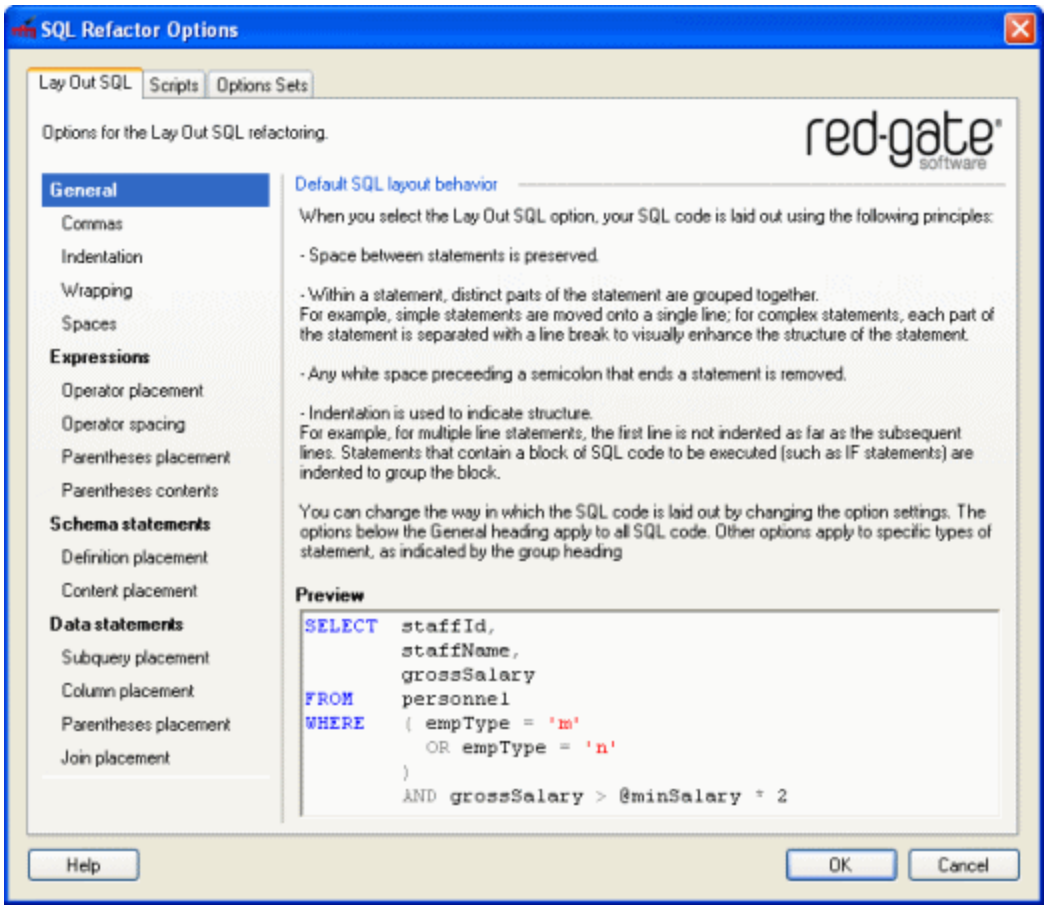
You can undo the changes to the script using the standard SQL Server Management Studio Undo features.

To keep the changes, save the script in the usual way.

Lay out SQL options

You can set a number of options to specify how the Lay out SQL refactoring reformats your SQL code.

To do this, on the **SQL Refactor** menu, click **Options**, and then select the **Lay Out SQL** tab.



When you change an option, the Preview pane shows the effect on an example script.

There is more information about the effect of individual options in the text within the graphical user interface.

You can save your options settings to an *options set* so that you can reuse them later. For example, you may wish to save a number of different options sets that you can apply in different circumstances. You can also reset your options to the SQL Refactor defaults. For details, see Options sets.

Default SQL layout behavior

When you select the **Lay Out SQL** option, your SQL code is laid out using the following principles:

- Space between statements is preserved.
- Within a statement, distinct parts of the statement are grouped together.

  For example, simple statements are moved onto a single line; for complex statements, each part of the statement is separated with a line break to visually enhance the structure of the statement.
- Any white space preceding a semicolon that ends a statement is removed.
- Indentation is used to indicate structure.

  For example, for multiple line statements, the first line is not indented as far as the subsequent lines. Statements that contain a block of SQL code to be executed (such as IF statements) are indented to group the block.

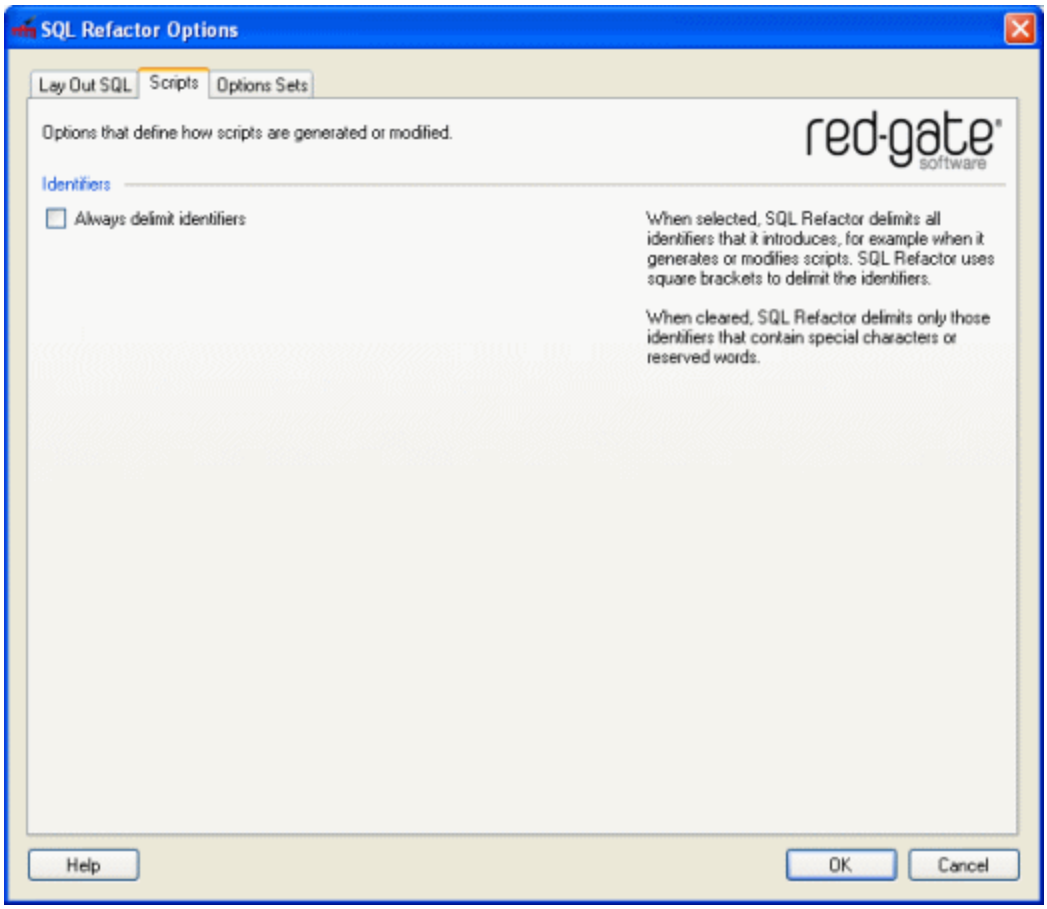
The options below the **General** heading apply to all SQL code. Other options apply to specific types of statement, as indicated by the group heading.

Script options

You can specify how SQL Refactor delimits identifiers that it introduces, for example when it generates script (such as when you use the Split Table refactoring), and when it modifies scripts (such as when you use Encapsulate As Stored Procedure).

SQL Refactor uses square brackets [ ] to delimit identifiers.

To set the option for delimiting identifiers, on the **SQL Refactor** menu, click **Options**, and then select the **Scripts** tab.



Select the **Always delimit identifiers** check box if you want SQL Refactor to delimit all identifiers that it introduces.

Clear the **Always delimit identifiers** check box if you want SQL Refactor to delimit only those identifiers that contain special characters or reserved words.

You can save your options settings to an *options set* so that you can reuse them later. For example, you may wish to save a number of different options sets that you can apply in different circumstances. You can also reset your options to the SQL Refactor defaults. For details, see Options sets.

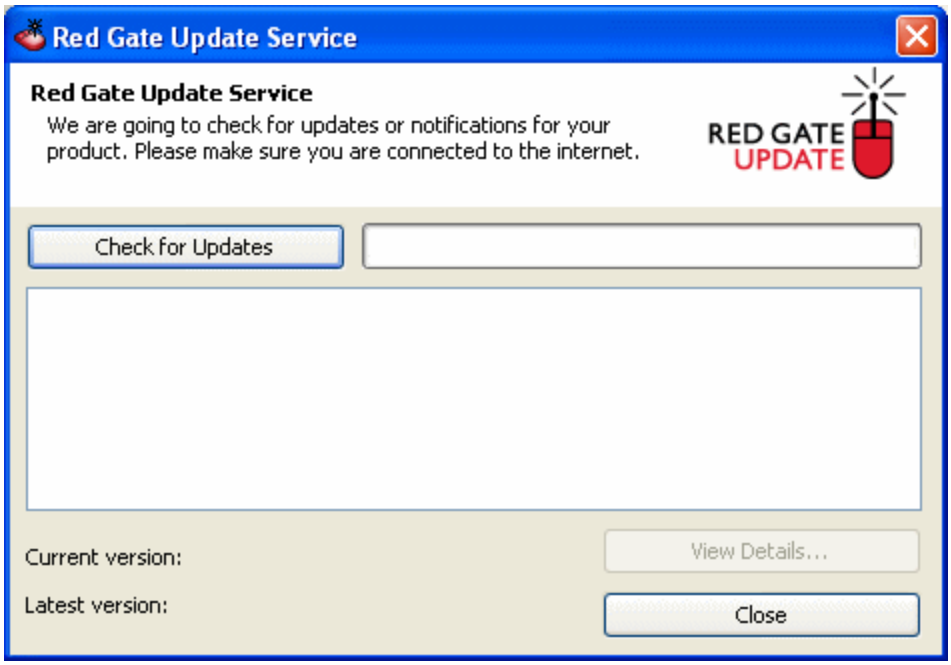
Check for Updates

Periodically, Red Gate Software issues updates for SQL Refactor. For example:

- free maintenance upgrades
- major upgrades that you can purchase
- notifications to inform you about new products or offers from Red Gate Software

To check for any updates that are available for download:

1. On the **SQL Refactor** menu, click **Check for Updates**.



2. Ensure that you are connected to the Internet, and then on the **Red Gate Update Service** dialog box, click **Check for Updates**.

   The updates that are available are listed.

3. Click **View Details**.

   Red Gate Update Service displays the available updates and notifications in your default Internet browser for you to download.

Uninstalling SQL Refactor

To uninstall SQL Refactor:

1. Use the Microsoft® Windows® Add or Remove Programs feature to remove the **SQL Refactor** program in the usual way.
2. In SQL Server Management Studio, on the **Tools** menu, click **Customize**.
3. Drag the **SQL Refactor** menu from the menu bar, or right-click on the **SQL Refactor** menu and click **Delete**.
4. In the **Customize** dialog box, click **Close**.

Acknowledgements

Copyright information

Trademarks and registered trademarks