erwin Data Modeler

Editing Forward Engineering Templates

Release 2020 R1

Legal Notices

This Documentation, which includes embedded help systems and electronically distributed materials (hereinafter referred to as the "Documentation"), is for your informational purposes only and is subject to change or withdrawal by erwin Inc. at any time. This Documentation is proprietary information of erwin Inc. and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of erwin Inc.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all erwin Inc. copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to erwin Inc. that all copies and partial copies of the Documentation have been returned to erwin Inc. or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, ERWIN INC. PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL ERWIN INC. BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF ERWIN INC. IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is erwin Inc.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19 (c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Contact erwin

Understanding your Support

Review support maintenance programs and offerings.

Registering for Support

Access the erwin support site and click Sign in to register for product support.

Accessing Technical Support

For your convenience, erwin provides easy access to "One Stop" support for all editions of erwin Data Modeler, and includes the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- erwin Support policies and guidelines
- Other helpful resources appropriate for your product

For information about other erwin products, visit http://erwin.com/products.

Provide Feedback

If you have comments or questions, or feedback about erwin product documentation, you can send a message to techpubs@erwin.com.

erwin Data Modeler News and Events

Visit www.erwin.com to get up-to-date news, announcements, and events. View video demos and read up on customer success stories and articles by industry experts.

Contents

Introduction

AllFusion erwin DM r7.0 introduced the Template Language component (TLX) as a new template/macro language. TLX is employed in a variety of places in the product where text expansion is necessary.

The Forward Engineering and Alter Script components use TLX in order to generate the necessary SQL statements. You can edit these Forward Engineering and Alter Script templates to provide custom SQL output using erwin DM r7.3 and later.

For the rest of this document, the term "templates" refers to the Forward Engineering and Alter Script templates.

Note: Templates were present in AllFusion erwin DM r7.0 through AllFusion erwin DM r7.2. However, as noted in the documentation of those releases, these templates used an interim format and changes to them were unsupported. Prior versions of templates cannot be used in erwin DM r7.3 and later. Both the file format and the template syntax have changed with the production version of templates. Changes to templates are supported. The file format and template syntax will be supported in future versions of the product and changes made will be usable when the software is upgraded.

This section contains the following topics

Documentation

For effective template editing, you need to be familiar with erwin DM's metamodel, and have a working knowledge of TLX.

The erwin DM metamodel is documented in the erwin Data Modeler Metamodel Reference bookshelf. The syntax for TLX and the macros available are documented in the *Template Language and Macro Reference*.

Template Location

By default, the templates are installed to a subdirectory of the Application Data directory of your machine, typically for a 32-bit version: C:\ProgramData\erwin\Data Modeler\9.8\Templates.

Note: If you do not use the *All Users* setting during product installation and enter a specific user instead, then the *All Users* subdirectory is the user name you entered.

The forward engineering template files have an extension of .fet and are named to correspond with the DBMS to which they apply. For example, the default template file for Oracle is *Oracle.fet*.

Note: You will find other files in the Templates directory that have different extensions. These are templates for other purposes and are not covered by this document.

If you want to restore the default template files, the install program places read-only copies of the files in the BackupFiles\Templates subdirectory where this product is installed.

Terminology

The terminology used differs slightly from that of older versions of erwin DM. The following table describes the differences in the terminology:

Term	Description
Template	A piece of TLX code. In this document, it refers specifically to templates used for the purpose of generating SQL for Forward Engineering or Alter Script. For example, the *Create View* template is used to generate a *CREATE VIEW* statement.
Template File	A collection of templates for a specific purpose stored externally. For example, the Oracle Template File (*Oracle.fet*) contains all the templates for Forward Engineering and Alter Script against an Oracle database.
Macro	A *function* supported by TLX that evaluates to a specific value or provides a specific control structure. For example, the *Date* macro evaluates to a string representing the current date/time.

Template Editing

This section provides an overview of template editing.

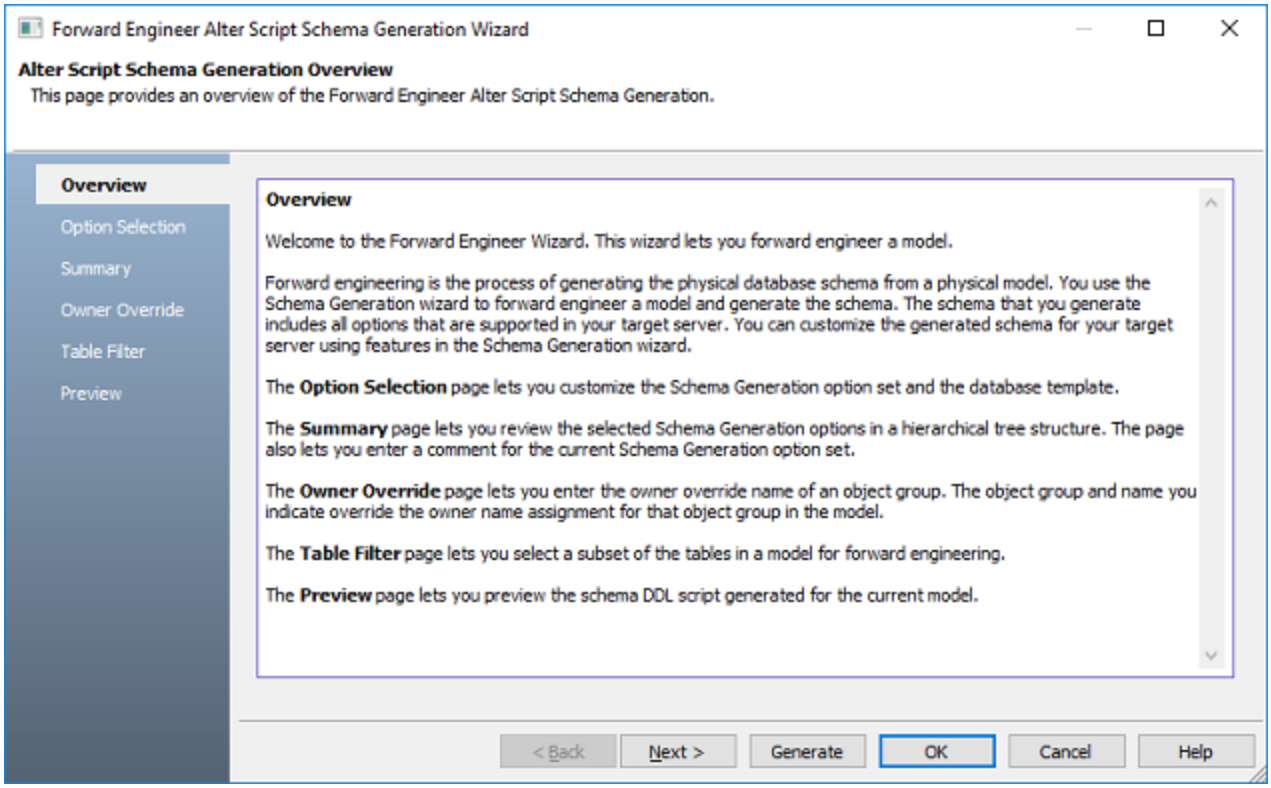
Opening the Template Editor

You can edit template files in any text editor that supports plain text format. However, the Template Editor in erwin DM provides several features that make editing templates easier, such as auto-expansion and macro selection.

To open the editor, click Forward Engineer, Forward Engineering Templates on the Actions menu.

This menu item is available whenever a physical or logical/physical model is active. The physical side of a logical/physical model does not need to be displayed in the workspace to edit its templates.

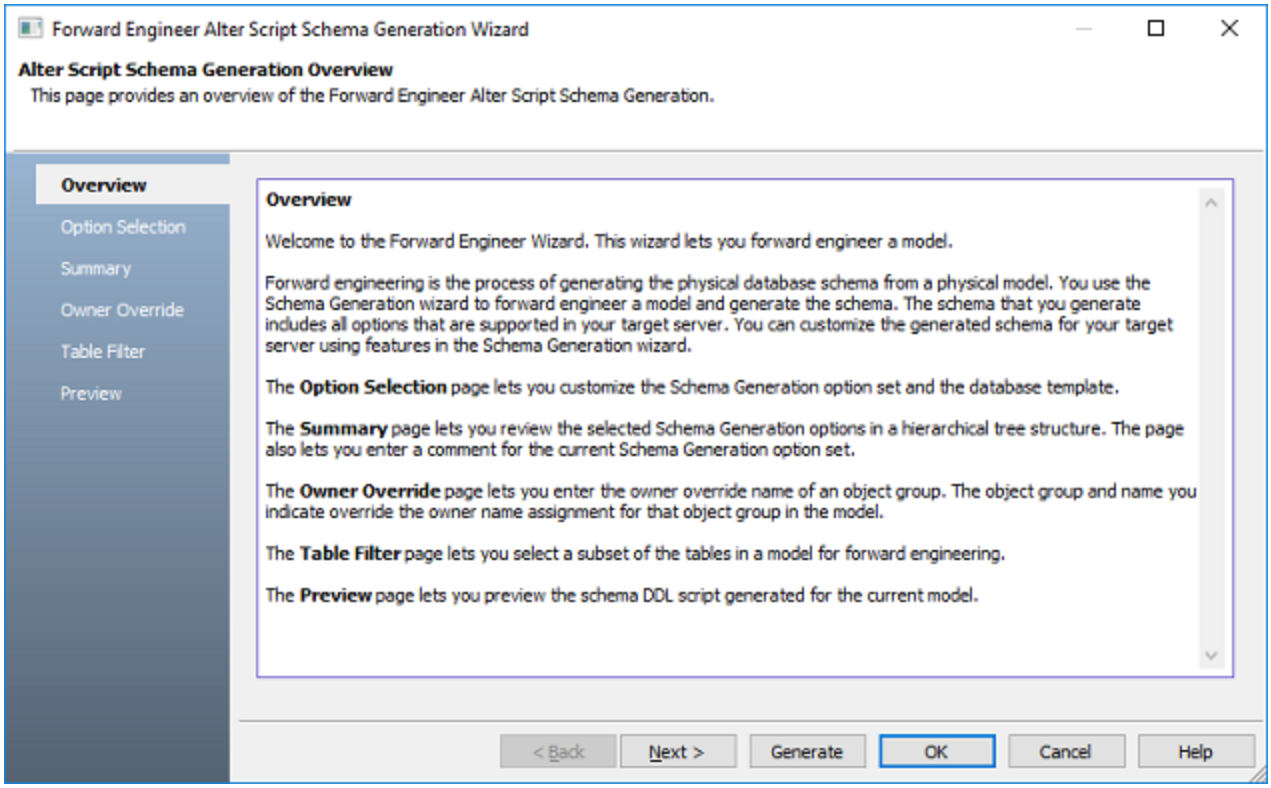
You can also open the editor when you click the Edit button in either the Forward Engineer Schema Generation or the Alter Script Schema Generation dialog, as shown in the following illustration:



Option Sets

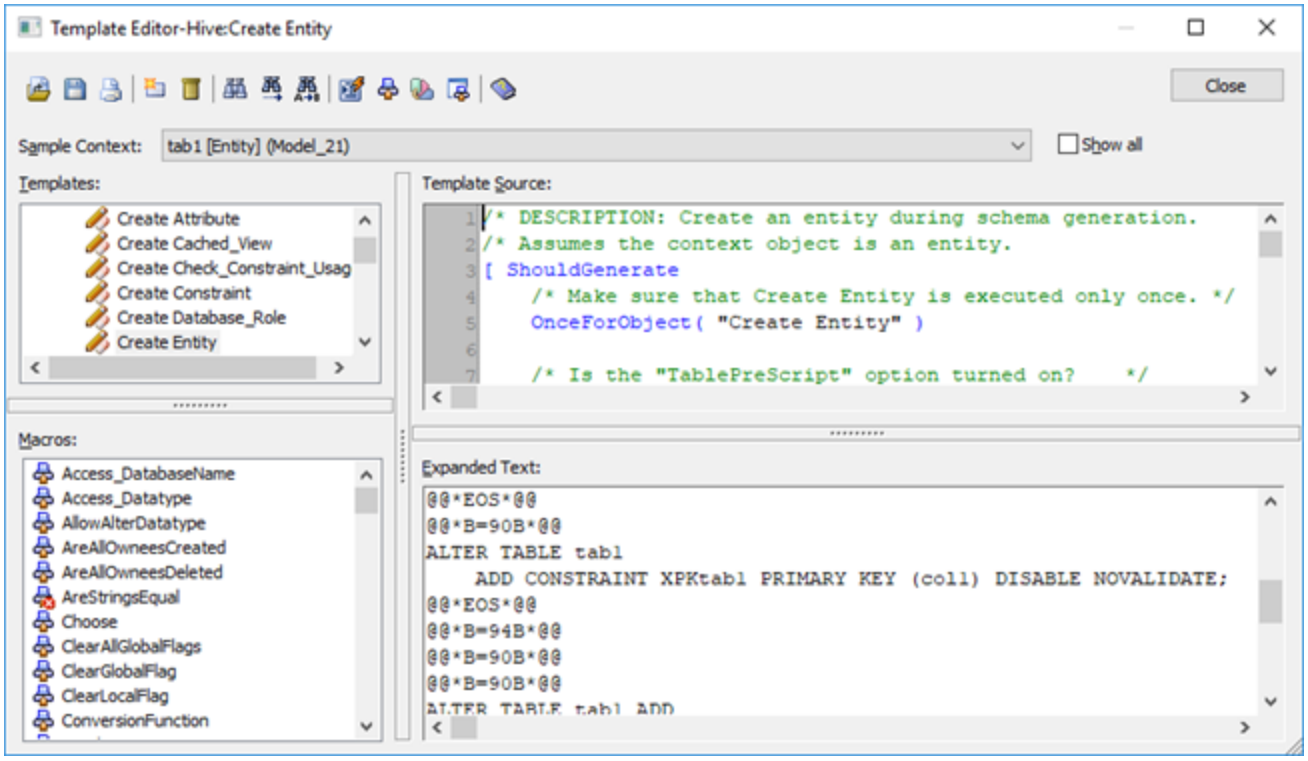
Each physical or logical/physical model has a *FE_Option_Set* object associated with it that holds the option settings you select. There is a default that is created by erwin DM and you can create new ones. Though several *FE_Option_Set* objects can exist in the model, only one can be designated as active at a time.

When the Template Editor expands a template, it takes into account the current settings for the model. In erwin DM, options are not saved to the model immediately upon selection in the user interface. To write the settings to the model and have them available to the Template Editor, you must either save them explicitly with the *Save* or *Save As* buttons, or save them implicitly when you click the *Generate* or *Preview* buttons, as shown in the following illustration:



The Template Editor

The Template Editor is shown in the following illustration:



The following list describes the tools and controls available in the Template Editor:

Opens a different template file.

Saves the template file or lets you make a copy of the file.

Prints the selected template.

Inserts a new template into the template file.

Marks or unmarks the selected template for deletion and deletes the template at the time of saving the template file.

Searches for text in the template.

Repeats your last search in the template.

Replaces text in the template.

Toggles auto-indenting in the template.

Toggles sorting of macros by categories.

Invokes the Template Editor Preferences dialog.

Invokes the Macro Categories dialog.

Launches the online help.

Sample Context

Allows you to select the starting object that will be used for the expansion of the template. By default, the editor attempts to select an *Entity* object when it is invoked. For each object listed, it shows the name of the object, its type, and a string representing its ownership chain in the model to allow objects with the same name to be distinguished from each other.

Note: Consult the *Template Language and Macro Reference* document to ensure you have a basic understanding of the context stack.

Since a large model might have sufficient objects to exceed Windows' limits for a combo box, only three objects of each type in the model are added to the list. If you want to see all objects in your model, select the *Show All* check box.

Templates

Shows all templates found in the template file. They are categorized based on their names as follows: *Create*, *Alter*, *Drop*, and *Insert* templates; *Clause* templates (reusable fragments employed by other templates); and *Miscellaneous* templates. Double-click a template to edit it. By default, when the Template Editor is invoked, it attempts to select the *Create Entity* template.

Macros

Shows the macros supported in TLX. By default, they are listed alphabetically. You can toggle the display to show them grouped by categories. You can create new categories and add macros to them using the *Macro Categories* dialog. The icon next to the macro indicates its deprecation status. You can learn about the various deprecation levels in the *Template Language and Macro Reference* document, and suppress the display of deprecated macros using the *Template Editor Preferences* dialog. Double-click a macro to insert it into the template at the current location.

Template Source

Shows the TLX code of the template. It is syntax-colored, but you can change the color choices in the *Template Editor Preferences* dialog.

Expanded Text

Shows the result of expanding the template against the current context object. Alternatively, it may show error information if a mistake is made in editing the template. It is syntax-colored, but you can change the color choices in the *Template Editor Preferences* dialog.

Template Editing Example

This section provides an example of creating a template to demonstrate the mechanics of editing a template. This example creates a very simple version of a *CREATE TABLE* template and tests it against the *EMPLOYEE* table in the *emovies.erwin* sample model.

Note: This example assumes the target server of your model is SQL Server 2000. However, the process is the same for all supported target servers.

The template produced here will not cover the full syntax of a *CREATE TABLE* statement. Once you understand the principles here, you can examine the full templates shipped with erwin DM to guide you in customizing your SQL.

Follow these steps to create the example template:

Basic Template - Version 1

The first pass demonstrates the steps to produce the following SQL:

```
create table EMPLOYEE
(
  employee_first_name varchar(20),
  employee_address varchar(20),
  employee_phone integer,
  employee_address_2 varchar(20),
  employee_number varchar(20),
  soc_sec_number integer
  hire_date datetime,
  salary integer,
  email varchar(20),
  store_number integer,
  supervisor varchar(20)

)
```

Getting Started

Template files are quite extensive in order to cover all of the SQL required for erwin DM's processes. The easiest way to start is to make a copy of an existing template file and edit it to produce the custom SQL you want:

- Go to the <install folder>\erwin\Data Modeler\[r*n*]\Templates directory, make a copy of *SqlServer.fet*, and name it *SqlServer2.fet*. This prevents unwanted changes to the real template file.
- Start erwin DM and load *emovies.erwin*.
- Click Forward Engineering Templates on the Tools menu to invoke the Template Editor.
- Click the *Open* tool and load *SqlServer2.fet*.
- Make sure the *EMPLOYEE [Entity]* table is selected in the *Sample Context* box. If you cannot find it, turn on *Show All*.

Entry Point Templates

The Forward Engineering component of erwin DM looks for templates with certain names to use as the starting point for a particular SQL command. For a *CREATE* statement, the name of the entry point template is *Create XXX* where *XXX* is the class name for the object type. The entry points for other types of statements follow this pattern; *DROP* statements have entry points of *Drop XXX*, *ALTER* statements have entry points of *Alter XXX*, and so on.

Note: Consult the *erwin Metamodel Reference* to locate the class names for various object and property types.

The class name for a table is *Entity*, so the *Create Entity* template is edited:

- Select the *Create Entity* template (it is probably already selected) and press F2. Rename that template to *Create Entity old*.
- Select the *New Template* tool and name the new template *Create Entity*.

At this point, the Template Editor attempts to expand the new template and displays a message that an undetermined parsing error occurred - this is expected.

The Template

This version of the template uses three macros: *Property*, *ForEachOwnee*, and *ListSeparator*.

Note: Consult the *Template Language and Macro Reference* document if you are uncertain of TLX syntax, or to see full descriptions of the macros used here.

- Type the following code into the *Template Source* field of the editor:

```
"create table " Property("Physical_Name")
"\n("
ForEachOwnee("Attribute")
{
  ListSeparator(",")
  "\n\t" Property("Physical_Name") " " Property("Physical_
Data_Type")
}
"\n)"
```

After a moment, the following text appears in the *Expanded Text* field:

```
create table EMPLOYEE
(
  employee_first_name varchar(20),
  employee_address varchar(20),
  employee_phone integer,
  employee_address_2 varchar(20),
  employee_number varchar(20),
  soc_sec_number integer
  hire_date datetime,
  salary integer,
  email varchar(20),
  store_number integer,
  supervisor varchar(20)

)
```

Splitting SQL Scripts - Version 2

The Forward Engineering component needs to be able to split up the various SQL statements that are generated to the script. Parsing the script during generation is too slow, so tokens are placed into the script to indicate the split points. These tokens are inserted by a macro called *FE::EndOfStatement*.

Tokens representing instructions to the Forward Engineering component are all delimited by double @ symbols (@@).

- Put your cursor in the *Template Source* field after the last line.
- Double-click the *FE::EndOfStatement* macro in the *Macros* tree.

After a moment, the following text appears in the *Expanded Text* field:

```
create table EMPLOYEE
(
  employee_first_name varchar(20),
  employee_address varchar(20),
  employee_phone integer,
  employee_address_2 varchar(20),
  employee_number varchar(20),
  soc_sec_number integer
  hire_date datetime,
  salary integer,
  email varchar(20),
  store_number integer,
  supervisor varchar(20)
)
go
```

@@*EOS*@@

Subsidiary Templates - Version 3

Describes subsidiary templates and the use of the *Execute* macro.

Execute Macro

One template can delegate some of the processing to another template using the *Execute* macro. Another template is created called *Emit FK* and it is delegated to produce the foreign key constraints for the table:

- Select the *New Template* tool and name the new template *Emit FK*.
- Add the following as temporary code for the new template. This code will be replaced later in the example.

```
"Got here"
```

- Double-click the *Create Entity* template again to go back to editing it and add the following text at the end. The use of the *Equal* macro causes nothing to emit if the current *Key_Group* is not a foreign key.

```
ForEachOwnee("Key_Group")
{
  Equal( Left( Property( "Key_Group_Type" ), 2 ), "IF" )
  Execute("Emit FK")
  FE::EndOfStatement
}
```

After a moment, the following text appears in the *Expanded Text* field:

```
create table EMPLOYEE
(
  employee_first_name varchar(20),
  employee_address varchar(20),
  employee_phone integer,
  employee_address_2 varchar(20),
  employee_number varchar(20),
  soc_sec_number integer
  hire_date datetime,
  salary integer,
  email varchar(20),
  store_number integer,
  supervisor varchar(20)
)
go
@@*EOS*@@
Got here
go
@@*EOS*@@
Got here
go
@@*EOS*@@
```

Controlling the Context Stack

The *Got here* statement can now be replaced in the *Emit FK* template with code to emit an *ALTER* statement that adds the foreign key. Since such a statement requires information from the *Entity* object, the *Attribute* objects owned by it, the *Relationship* object, and the *Key_Group_Member* objects, the template code will have to control the context stack to make sure that properties are read from the correct object.

Note: This is not the most efficient way to produce the intended output, but it accomplishes the task without complicating the example with a lot of new macros.

- Double-click on the *Emit FK* template to edit it.
- Replace the existing text with the following code:

```
"alter table " PushOwner Property("Physical_Name") Pop
"\nadd foreign key ("
ForEachOwnee("Key_Group_Member")
{
  ListSeparator(",") Property("Physical_Name")
}
") references "
PushReference("Relationship_Ref")
PushReference("Parent_Entity_Ref")
Property("Physical_Name")
Pop
Pop
" ("
ForEachMigratingColumn
{
  ListSeparator(",") Property("Physical_Name")
}
")"
PushReference("Relationship_Ref")
"\n\ton delete "
UpperCase( Property("Parent_Delete_Rule") )
Pop
```

- Ignore the parsing error that appears in the *Expanded Text* field (because the template was not entered from the correct starting point) and double-click the *Create Entity* template to get the entry point template to evaluate.

After a moment, the following text appears in the *Expanded Text* field:

```
create table EMPLOYEE
(
  employee_first_name varchar(20),
  employee_address varchar(20),
  employee_phone integer,
  employee_address_2 varchar(20),
  employee_number varchar(20),
  soc_sec_number integer
  hire_date datetime,
  salary integer,
  email varchar(20),
  store_number integer,
  supervisor varchar(20)
)
go
@@*EOS*@@
alter table EMPLOYEE
add foreign key (store_number) references STORE (store_number)
on delete NO ACTION
go
@@*EOS*@@
alter table EMPLOYEE
add foreign key (supervisor) references EMPLOYEE (employee_number)
on delete NO ACTION
go
@@*EOS*@@
```

Sorting the Output - Version 4

One of the foreign keys in the *EMPLOYEE* table is self-referential. However, the other is not; it references the *STORE* table. If the order of processing the *Entity* objects results in the *STORE* table emitting after the *EMPLOYEE* table, the *ALTER* statement creating the foreign key will fail.

To avoid this, tokens can be emitted that instruct the Forward Engineering component to sort types of statements into groups. These groups are called *buckets*. Any statement placed in Bucket #1 is emitted before any statement placed in Bucket #2, which is before any statement in Bucket #3, and so on. There can be an arbitrary number of buckets. The macro *FE::Bucket* inserts the bucket tokens.

- Insert the following code as the first line of the *Create Entity* template:

  ```
  FE::Bucket("10")
  ```

- Insert the following code right after the first occurrence of the *FE::EndOfStatement* macro:

  ```
  FE::Bucket("20")
  ```

After a moment, the following text appears in the *Expanded Text* field:

```
@@*B=10*@@
create table EMPLOYEE
(
  employee_first_name varchar(20),
  employee_address varchar(20),
  employee_phone integer,
  employee_address_2 varchar(20),
  employee_number varchar(20),
  soc_sec_number integer
  hire_date datetime,
  salary integer,
  email varchar(20),
  store_number integer,
  supervisor varchar(20)
)
go
@@*EOS*@@
@@*B=20*@@
alter table EMPLOYEE
add foreign key (store_number) references STORE (store_number)
on delete NO ACTION
go
@@*EOS*@@
```

When the templates are executed against the entire model, all of the *CREATE TABLE* statements (Bucket #10) will emit before any of the *ALTER* statements (Bucket #20).