Example SQL code that builds a small database with only 5 tables:

- *tblPRODUCT_TYPE*
- *tblPRODUCT*
- *tblCUSTOMER*
- *tblORDER*
- *tblORDER_PRODUCT*

Lines #44 through #48 have foreign keys defined using the ALTER TABLE statements

```
3  CREATE TABLE tblPRODUCT_TYPE
4  (ProductTypeID INT IDENTITY(1,1) primary key,
5  ProductTypeName varchar(50) not null,
6  ProductTypeDescr varchar(500) NULL)
7  GO
8
9  CREATE TABLE tblPRODUCT
10 (ProductID INT IDENTITY(1,1) PRIMARY KEY,
11 ProductName varchar(50) NOT NULL,
12 Price Numeric(8,2) not null,
13 ProductTypeID INT FOREIGN KEY REFERENCES tblPRODUCT_TYPE
     (ProductTypeID) NOT NULL,
14 ProductDescr varchar(500) NULL)
15 GO
16
17 CREATE TABLE tblCUSTOMER
18 (CustomerID INT IDENTITY(1,1) PRIMARY KEY,
19 Fname varchar(30) not null,
20 Lname varchar(30) not null,
21 BirthDate Date not null)
22 GO
23
24 CREATE TABLE tblORDER
25 (OrderID INT IDENTITY(1,1) primary key,
26 OrderDate Date not null,
27 CustomerID INT FOREIGN KEY REFERENCES tblCUSTOMER (CustomerID) not
     null)
28 GO
29
```

```
30  CREATE TABLE tblORDER_PRODUCT
31  (OrderProductID INT IDENTITY(1,1) primary key,
32  OrderID INT not null,
33  ProductID INT not null,
34  Quantity INT not null)
35  GO
36
37  ALTER TABLE tblORDER_PRODUCT
38  ADD CONSTRAINT FK_tblOrder_Product_OrderID
39  FOREIGN KEY (OrderID)
40  REFERENCES tblORDER (OrderID)
41  GO
42
```

```
43
44  ALTER TABLE tblORDER_PRODUCT
45  ADD CONSTRAINT FK_tblOrder_Product_ProductID
46  FOREIGN KEY (ProductID)
47  REFERENCES tblPRODUCT (ProductID)
48  GO
49
50
```

Next, lines 51 through 63 are brief INSERT INTO statements for tables that do not have any foreign keys. We MUST populate these tables first!

```
51  INSERT INTO tblPRODUCT_TYPE (ProductTypeName, ProductTypeDescr)
52  VALUES
53  ('Fruit', 'Sweet, juicey, and grows on a tree'),
54  ('Meat','Tough, full of protein...used to be an animal'),
55  ('Vegetable', 'Not as sweet, nutritious and grows in the dirt'),
56  ('Clothing', 'Anything people wear')
57
58
59
60  INSERT INTO tblCUSTOMER (Fname, Lname, BirthDate)
61  VALUES ('Kenny', 'TheCat','October 20, 2012'), ('Mitch','TheCat',
      'November 5, 2011')
62
63  GO
64
```

In lines 65 through 71 we define a stored procedure that has two parameters:

- *@PTName_2*
- *@PTID_OUT*

The first parameter is an 'input' and participates in the lookup of the foreign key identifier. The FK value is then handed off to the second parameter which is an 'output' and can be seen by other stored procedures. This 'hand-off' process greatly improves the speed at which foreign keys are found during high-volume insert processes (such as Starbucks Coffee needing to find customerID or productID 30 million times a day).

```
65  -- this is the second stored procedure --> nested look-up
66  CREATE PROCEDURE gthay_NewNestedSPROC
67  @PTNAME_2 varchar(50),
68  @PTID_OUT INT OUTPUT
69  AS
70  SET @PTID_OUT = (SELECT ProductTypeID FROM tblPRODUCT_TYPE WHERE
      ProductTypeName = @PTName_2)
71  GO
72
73
74
75
```

Lines 79 through 109 are the definition of a stored procedure. As a reminder, stored procedures are used to conduct transactions (INSERT, UPDATE, and DELETE statements) and increase the volume and security of these critical events.

```
79  CREATE or ALTER PROCEDURE gthay_New_Procedure1
80  @PTName varchar(50),
81  @Prod varchar(50),
82  @prodPrice numeric(8,2),
83  @Pdescr varchar(500)
84  AS
85
86  DECLARE @PTID INTEGER
87
```

Lines 80 through 83 represent 4 parameters.

These parameters will be used inside the INSERT statement as seen in line 100 as well as during the calling of the nested procedure in lines 89.

As a reminder, a parameter is an externally visible memory object that allows an application to communicate with a database. This is how data is presented to an INSERT statement to be placed into a database table during a transaction.

```
88  EXEC gthay_NewNestedSPROC
89  @PTNAME_2 = @PTName,
90  @PTID_OUT = @PTID OUTPUT
91
92  IF @PTID IS NULL
93      BEGIN
94          PRINT 'Hey...@PTID is empty...check spelling';
95          THROW 55667, '@PTID cannot be NULL; process is
              terminating',1;
96      END
97
```

Line 92 through 96 above include several keywords that are considered 'Control of Flow' language.

- *IF*
- *BEGIN*
- *END*

Lines 98 - 109 have an 'explicit' transaction. An explicit transaction allows for multiple INSERT, UPDATE, and DELETE statements to be bundled together into a single transaction.

```
98  BEGIN TRANSACTION T1
99  INSERT INTO tblPRODUCT (ProductName, ProductTypeID, Price,
      ProductDescr)
100 VALUES (@Prod, @PTID, @prodPrice, @Pdescr)
101
102 IF @@ERROR <> 0
103     BEGIN
104         PRINT 'Uhhhh....something broke during the transaction'
105         ROLLBACK TRANSACTION T1
106     END
107 ELSE
108     COMMIT TRANSACTION
109 GO
110
```

Next is some example SQL code that calls the stored procedure with data to be entered into the database. The keyword used to call the stored procedure is 'EXECUTE'

```
111
112   EXECUTE gthay_New_Procedure1
113   @PTName = 'Clothing',
114   @Prod = 'All Black Jordans',
115   @prodPrice = 287.39,
116   @Pdescr = 'Black high-tops'
117
```