Module 4: Relationships

Lab: Planning and Implementing Referential Integrity

Exercise 1: Planning Referential Integrity

Task 1: Prepare the Lab Environment

1. Ensure that the 10985C-MIA-DC and 10985C-MIA-SQL virtual machines are both running, and then log on to 10985C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
2. In the **D:\Labfiles\Lab04\Starter** folder, right-click **Setup.cmd**, click **Run as Administrator**, and then wait for setup to complete.

Task 2: Identify Relationships

1. On the taskbar, click **Microsoft SQL Server Management Studio**, in the **Connect to Server** dialog box, in the **Server type** field, select **Database Engine**, in the **Server name** field, type **MIA-SQL**, in the **Authentication** field, select **Windows Authentication**, and then click **Connect**.
2. In Object Explorer, expand **Databases**, expand **OrdersDatabase**, right-click **Database Diagrams**, and then click **New Database Diagram**.
3. If a dialog box appears asking if you wish to create support objects for database diagramming, click **Yes**.
4. In the **Add Table** dialog box, press and hold down the SHIFT key, click **Products**, click **Add**, and then click **Close**.
5. Rearrange the tables so that you can see them all at the same time. You can do this by clicking the table header and dragging the table to the required place.
6. Review the database diagram and determine the relationships between the tables and what types of relationships they are (one-to-one, one-to-many, or many-to-many).

Task 3: Plan Foreign Keys

1. Review the tables in the diagram again, noting the columns that are defined as primary keys, and that there are no foreign keys defined.
2. Use the business rules to decide which foreign keys you should create to enforce referential integrity. Consider which columns should have a foreign key, and which column each foreign key should reference.
3. On the **File** menu, click **Save Diagram_0**.
4. In the **Choose Name** dialog box, type **Keys**, and then click **OK**.
5. Close the database diagram pane.

Exercise 2: Implementing Referential Integrity by Using Constraints

Task 1: Implement a Foreign Key in the Orders Table

> **Note:** You can find the completed Transact-SQL statements for this exercise in the **Create ForeignKeys.sql** file in the **D:\Labfiles\Lab04\Solution** folder.

1. In SQL Server Management Studio, click **New Query**.
2. In the query window, type the following Transact-SQL statements, and then click **Execute**:

```
USE OrdersDatabase;
GO

ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Customers FOREIGN KEY (CustomerID)
REFERENCES Customers (CustomerID);
GO
```

3. To test the foreign key, in the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

```
INSERT INTO Orders
VALUES (105,2, GETDATE());
GO
```

   The INSERT is successful.

4. To test the foreign key again, in the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

```
INSERT INTO Orders
VALUES (106,5, GETDATE());
GO
```

   The INSERT is prevented by the foreign key constraint because the **CustomerID** value **5** does not exist in the **Customers** table.

Task 2: Implement a Foreign Key in the CustomerDetails Table

1. In the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

```
ALTER TABLE CustomerDetails
ADD CONSTRAINT FK_CustomerDetails_Customers FOREIGN KEY (CustomerID)
REFERENCES Customers (CustomerID);
GO
```

2. To test the foreign key, in the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

```
INSERT INTO CustomerDetails
VALUES (5,'9832 Mt. Dias Blv.', 'Chicago','97321', '08/09/1970');
GO
```

The INSERT is prevented by the foreign key constraint because the **CustomerID** value **5** does not exist in the **Customers** table.

Task 3: Implement Foreign Keys in the LineItems Table

1. In the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

   ```
   ALTER TABLE LineItems
   ADD CONSTRAINT FK_LineItems_Orders FOREIGN KEY (OrderID)
   REFERENCES Orders (OrderID);
   GO
   ```

2. To test the foreign key, in the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

   ```
   INSERT INTO LineItems
   VALUES (101,33,30.00,1);
   GO
   ```

   The INSERT is successful.

3. To test the foreign key again, in the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

   ```
   INSERT INTO LineItems
   VALUES (106,44,30.00,1);
   GO
   ```

   The INSERT is prevented because the **OrderID** value **106** does not exist in the **Orders** table.

4. In the query window under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

   ```
   ALTER TABLE LineItems
   ADD CONSTRAINT FK_LineItems_Products FOREIGN KEY (ProductID)
   REFERENCES Products (ProductID);
   GO
   ```

5. To test the foreign key, in the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

   ```
   INSERT INTO LineItems
   VALUES (102,22,15.00,1);
   GO
   ```

The INSERT is successful.

6. To test the foreign key again, in the query window under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

```
INSERT INTO LineItems
VALUES (104,66,30.00,1);
GO
```

The INSERT is prevented because the **ProductID** value **66** does not exist in the **Products** table.

7. On the **File** menu, click **Save SQLQuery1.sql**.

8. In the **Save File As** dialog box, browse to **D:\Labfiles\Lab04\Starter**, in the **File name** box, type **CreateForeignKeys**, and then click **Save**.

Exercise 3: Implementing Cascading Referential Integrity

Task 1: Configure Cascading Referential Integrity in the CustomerDetails Table

> **Note:** You can find the completed Transact-SQL statements for this exercise in the **ImplementCascadingIntegrity.sql** file in the **D:\Labfiles\Lab04\Solution** folder.

1. In SQL Server Management Studio, click **New Query**, in the query window, type the following Transact-SQL statements, and then click **Execute**:

```
USE OrdersDatabase;
GO

DELETE Customers
WHERE CustomerID = 2;
GO
```

2. Review the results, and note that the DELETE failed because of a foreign key constraint.

3. In the query window, under the existing Transact-SQL statements, type the following Transact-SQL statements, select them, and then click **Execute**:

```
ALTER TABLE CustomerDetails
DROP CONSTRAINT FK_CustomerDetails_Customers;
GO

ALTER TABLE CustomerDetails
ADD CONSTRAINT FK_CustomerDetails_Customers FOREIGN KEY (CustomerID)
REFERENCES Customers (CustomerID) ON DELETE CASCADE;
GO
```

4. Select the Transact-SQL DELETE statement you typed in step 1, and then click **Execute**.

5. Review the results, and note that the DELETE failed again, because of the foreign key constraint on the **Orders** table.

Task 2: Configure SET DEFAULT in the Orders Table

1. In the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, which retrieves all orders from the **Orders** table. Select the statement, and then click **Execute**:

   ```
   SELECT * FROM Orders;
   GO
   ```

   Verify that customer 2 has several orders.

2. In the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, which creates a default constraint on the **CustomerID** column in **Orders** with a value of **0**. Select the statement, and then click **Execute**:

   ```
   ALTER TABLE Orders
   ADD CONSTRAINT DEF_CustomerID
   DEFAULT 0 FOR CustomerID;
   GO
   ```

3. Under the existing Transact-SQL statements, type the following Transact-SQL statement, which adds a row to the **Customers** table with a **CustomerID** value of **0**. Select the statement, and then click **Execute**:

   ```
   INSERT INTO Customers
   VALUES(0, 'Not Applicable', 'Not Applicable');
   GO
   ```

4. Under the existing Transact-SQL statements, type the following Transact-SQL statements, select them, and then click **Execute**:

   ```
   ALTER TABLE Orders
   DROP CONSTRAINT FK_Orders_Customers;
   GO

   ALTER TABLE Orders
   ADD CONSTRAINT FK_Orders_Customers FOREIGN KEY (CustomerID)
   REFERENCES Customers (CustomerID) ON DELETE SET DEFAULT;
   GO
   ```

5. Select the Transact-SQL DELETE statement you typed in step 1 of the previous task, and then click **Execute**.

6. Review the results, and note that the DELETE succeeded.

7. Select the Transact-SQL statement that you typed in Step 1, and then click **Execute**. This statement retrieves the details of all orders.

8. Review the results, and note that **OrderID** 101 now has a **CustomerID** value of 0.

9. Under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

```
SELECT * FROM CustomerDetails;
GO
```

10. Review the results, and note that **CustomerID** 2 has been removed.
11. On the **File** menu, click **Save SQLQuery2.sql**.
12. In the **Save File As** dialog box, browse to **D:\Labfiles\Lab04\Starter**, in the **File name** box, type **ImplementCascadingIntegrity**, and then click **Save**.
13. Close SQL Server Management Studio.

Lab Review

Question: Do you think that it was a good idea to implement the ON DELETE CASCADE and the ON DELETE SET DEFAULT options in the final exercise in the lab? What problems might this potentially cause? What might you have done instead to prevent these problems?

Answer Cascading the delete to the CustomerDetails table removes all customer information for the deleted customers, which means that it is more difficult to perform data analysis later on, when it is based on customer demographics. Using the SET NULL option in Orders results in order rows that are not associated with a specific customer, but instead have a default catch-all CustomerID value. Removing the context for an order in this way can make it very difficult to aggregate data in a meaningful manner, which again can reduce the usefulness of the data for analytics.

One possible solution would be to use triggers to make copies of the data before deleting it from the tables. This data could then be used for analytical purposes. If it is necessary to eliminate data that identifies a customer, you could selectively save the non-identifying elements, such as the city where the customer lives, or their age, and not save the identifying data, such as their name, phone number, or street address.