# Module 4: Relationships

# Lab: Planning and Implementing Referential Integrity

### Scenario

You work for an organization called Adventure Works, which is a retailer of bicycles and associated products. You are planning referential integrity for a section of a database called **OrdersDatabase**, an OLTP database that tracks customers, the orders that they place, and the products that are included in the orders. To implement referential integrity, you will use foreign keys. First, you will plan the foreign keys that you will need to enforce integrity; you will then implement the foreign keys; and finally, you will implement cascading referential integrity.

### Objectives

After completing this lab, you will have:

- Planned referential integrity for an OLTP database.
- Implemented referential integrity by using foreign keys.
- Implemented cascading referential integrity.

### Lab Setup

Estimated Time: 60 minutes

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**.

## Exercise 1: Planning Referential Integrity

### Scenario

You have established that the following business rules must be enforced:

- Every order must be associated with a specific customer.
- Every line item must be associated with a product and a valid order.
- Customer details must be associated with a valid customer.

In this exercise, you will use a database diagram to help you to plan which foreign key constraints you will need to enforce these rules.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Identify Relationships
3. Plan Foreign Keys

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. Browse to the **D:\Labfiles\Lab04\Starter folder**, and then run **Setup.cmd** as Administrator.

Task 2: Identify Relationships

1. Open SQL Server Management Studio and connect to the MIA-SQL Database Engine instance by using Windows Authentication.
2. In the **OrdersDatabase** database, create a database diagram, and then add the following tables to it:
    - **CustomerDetails**
    - **Customers**
    - **Orders**
    - **Products**
    - **LineItems**
3. 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. Use the database diagram and the business rules to decide which foreign key constraints you need to create to enforce referential integrity. Include in your considerations which columns should have a foreign key, and which column each foreign key should reference.
2. After planning the foreign keys, save the database diagram as **Keys**.
3. Close the database diagram pane.

> **Results**: In this exercise, you identified the keys required to enforce referential integrity rules.

# Exercise 2: Implementing Referential Integrity by Using Constraints

## Scenario

The following relationships exist in the database:

- A one-to-one relationship between **Customers** and **CustomerDetails**.
- A one-to-many relationship between **Customers** and **Orders**.
- A one-to-many relationship between **Orders** and **LineItems**.
- A one-to-many relationship between **Products** and **LineItems**.

To implement the required referential integrity, you need to create the following foreign key constraints:

- A foreign key on **OrderID** in **LineItems** that references **OrderID** in **Orders**.
- A foreign key on **ProductID** in **LineItems** that references **ProductID** in **Products**.
- A foreign key on **CustomerID** in **Orders** that references **CustomerID** in **Customers**.

- A foreign key on **CustomerID** in **CustomerDetails** that references **CustomerID** in **Customers**.

In this exercise, you will implement these foreign key constraints.

The main tasks for this exercise are as follows:

1. Implement a Foreign Key in the Orders Table
2. Implement a Foreign Key in the CustomerDetails Table
3. Implement Foreign Keys in the LineItems Table

Task 1: Implement a Foreign Key in the Orders Table

1. In a query window in SQL Server Management Studio, write a Transact-SQL ALTER TABLE statement to create a foreign key constraint called **FK_Orders_Customers** on the **CustomerID** column in the **Orders** table that references the **CustomerID** column in the **Customers** table. Do not include an ON UPDATE or ON DELETE clause.
2. To test the foreign key, write a Transact-SQL statement INSERT statement to add a row to the **Orders** table using the following values:
    - **OrderID** = **105**
    - **CustomerID** = **2**
    - **OrderDate** = **GETDATE()**
3. This INSERT statement should succeed.
4. To test the foreign key again, write a Transact-SQL statement INSERT statement to add a row to the **Orders** table using the following values:
    - **OrderID** = **106**
    - **CustomerID** = **5**
    - **OrderDate** = **GETDATE()**
5. This INSERT statement should be 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 same query window, write a Transact-SQL ALTER TABLE statement to create a foreign key constraint called **FK_CustomerDetails_Customers** on the **CustomerID** column in the **CustomerDetails** table that references the **CustomerID** column in the **Customers** table. Do not include an ON UPDATE or ON DELETE clause.
2. To test the foreign key, write a Transact-SQL statement INSERT statement to add a row to the **CustomerDetails** table using the following values:
    - **CustomerID** = **5**
    - **Address** = **'9832 Mt. Dias Blv.'**
    - **City** = **'Chicago'**
    - **Postal Code = '97321'**

- **DateOfBirth** = **'08/09/1970'**

This INSERT statement should be 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 same query window in SQL Server Management Studio, write a Transact-SQL ALTER TABLE statement to create a foreign key constraint called **FK_LineItems_Orders** on the **OrderID** column in the **LineItems** table that references the **OrderID** column in the **Orders** table. Do not include an ON UPDATE or ON DELETE clause.
2. To test the foreign key, write a Transact-SQL statement INSERT statement to add a row to the **LineItems** table using the following values:
    - **OrderID** = **101**
    - **ProductID** = **33**
    - **UnitPrice** = **30.00**
    - **Quantity** = **1**

    This INSERT statement should succeed.
3. To test the foreign key again, write a Transact-SQL statement INSERT statement to add a row to the **LineItems** table using the following values:
    - **OrderID = 106**
    - **ProductID = 44**
    - **UnitPrice = 30.00**
    - **Quantity = 1**

    This INSERT statement should be prevented by the foreign key constraint because the **OrderID** value **106** does not exist in the **Orders** table.
4. In the same query window in SQL Server Management Studio, write a Transact-SQL ALTER TABLE statement to create a foreign key constraint called **FK_LineItems_Products** on the **ProductID** column in the **LineItems** table that references the **ProductID** column in the **Products** table. Do not include an ON UPDATE or ON DELETE clause.
5. To test the foreign key, write a Transact-SQL statement INSERT statement to add a row to the **LineItems** table using the following values:
    - **OrderID** = **102**
    - **ProductID** = **22**
    - **UnitPrice** = **15.00**
    - **Quantity** = **1**

    This INSERT statement should succeed.
6. To test the foreign key again, write a Transact-SQL statement INSERT statement to add a row to the **LineItems** table using the following values:

- **OrderID = 104**
- **ProductID = 66**
- **UnitPrice = 30.00**
- **Quantity = 1**

This INSERT statement should be prevented by the foreign key constraint because the **ProductID** value **66** does not exist in the **Products** table.

7. Close the query window and save the file as **CreateForeignKeys.sql** in the **D:\Labfiles\Lab04\Starter** folder.

> **Results**: In this exercise, you implemented referential integrity in the **OrdersDatabase** database using constraints.

## Exercise 3: Implementing Cascading Referential Integrity

### Scenario

An additional business rule states that whenever a **CustomerID** is removed from the **Customers** table, which sometimes happens when a customer requests that their data is removed from the database, all associated information should also be removed, including customer details. However, the **Orders** table references the **CustomerID** column in the **Customers** table, and you do not want to cascade deletions from **Customers** to **Orders** because the order information is important for accounting and reporting purposes. Instead, you must ensure that when a row is deleted in **Customers**, the **CustomerID** value in **Orders** updates to a default value. In this exercise, you will configure referential integrity to implement these rules.

The main tasks for this exercise are as follows:

1. Configure Cascading Referential Integrity in the CustomerDetails Table
2. Configure SET DEFAULT in the Orders Table

Task 1: Configure Cascading Referential Integrity in the CustomerDetails Table

1. In SQL Server Management Studio, in a new query window, type and execute a Transact-SQL statement to delete the row from the **Customers** table that has the **CustomerID** value **2**. The delete should fail because of a foreign key constraint.
2. Type and execute an ALTER TABLE Transact-SQL statement to drop the **FK_CustomerDetails_Customers** constraint in the **CustomerDetails** table.
3. Type and execute an ALTER TABLE Transact-SQL statement to create a new foreign key constraint called **FK_CustomerDetails_Customers** on the **CustomerDetails** table. Include a clause to cascade delete actions.
4. Attempt to delete the row from the **Customers** table that has the **CustomerID** value **2** again. The delete should fail again because of the foreign key constraint in the **Orders** table.

Task 2: Configure SET DEFAULT in the Orders Table

1. Review the data in the **Orders** table and note that there are existing orders for customer 2.
2. In the same Transact-SQL query window, type and execute an ALTER TABLE Transact-SQL statement that creates a default constraint on the **CustomerID** column in **Orders**, with a

value of **0**.

3. Type and execute a Transact-SQL statement that adds a row to the **Customers** table using the following values:
    - **CustomerID** = **0**
    - **FirstName** = **'Not Applicable'**
    - **LastName** = **'Not Applicable'**
4. Type and execute an ALTER TABLE Transact-SQL statement to drop the **FK_Orders_Customers** constraint in the **Orders** table.
5. Type and execute an ALTER TABLE Transact-SQL statement to create a new foreign key constraint called **FK_Orders_Customers** on the **Orders** table. Include a clause to set the value to default for delete actions.
6. Attempt to delete the row from the **Customers** table that has the **CustomerID** value **2** again. The delete should succeed.
7. Check that the Order 101 now has a **CustomerID** value of 0.
8. Save the query file as **ImplementCascadingIntegrity.sql** in the **D:\Labfiles\Lab04\Starter** folder.
9. Close SQL Server Management Studio.

> **Results**: In this exercise, you implemented cascading referential integrity.

## 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?