Lab Manual
T2762-1, Developing SQL Databases, Part 1

Lab 2: Designing and Implementing Tables

Scenario

A business analyst from your organization has given you a draft design for some new tables being added to a database. You need to provide an improved schema design, based on good design practices. After you have designed the schema and tables, you need to implement them in the TSQL database.

Objectives

After completing this lab, you will be able to:

- Choose an appropriate level of normalization for table data.
- Create a schema.
- Create tables.

Virtual machine: **20762C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Designing Tables

Scenario

A business analyst from your organization has given you a first pass at a schema design for some new tables being added to the TSQL database. You need to provide an improved schema design, based on good design practices and an appropriate level of normalization. The business analyst was also confused about when data should be nullable. You need to decide about nullability for each column in your improved design.

The main tasks for this exercise are as follows:

1. Prepare the Environment
2. Review the Design
3. Improve the Design

▶ Task 1: Prepare the Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab02\Starter** folder as Administrator.

▶ Task 2: Review the Design

1. Open the **Schema Design for Marketing Development Tables.docx** from the **D:\Labfiles\Lab02\Starter** folder.
2. Review the proposed structure for the new tables.

▶ Task 3: Improve the Design

1. Complete the **Allow Nulls?** column for each table.
2. Save your document.

3. Review the suggested solution in **Schema Design for Marketing Development Tables.docx** in the **D:\Labfiles\Lab02\Solution** folder.
4. Close WordPad.

Results: After completing this exercise, you will have an improved schema and table design.

Exercise 2: Creating Schemas

Scenario

The new tables will be isolated in their own schema. You need to create the required schema called DirectMarketing and assign ownership to the dbo user.

The main tasks for this exercise are as follows:

1. Create a Schema

▶ Task 1: Create a Schema

1. Using SSMS, connect to MIA-SQL using Windows Authentication.
2. Open **Project.ssmssln** from the **D:\Labfiles\Lab02\Starter\Project** folder.
3. In the **Lab Exercise 2.sql** file, write and execute a query to create the **DirectMarketing** schema, and set the authorization to the **dbo** user.

Results: After completing this exercise, you will have a new schema in the database.

Exercise 3: Creating Tables

Scenario

You need to create the tables that you designed earlier in this lab. You should use appropriate nullability for each column and each table should have a primary key. At this point, there is no need to create CHECK or FOREIGN KEY constraints.

The main tasks for this exercise are as follows:

1. Create the Competitor Table
2. Create the TVAdvertisement Table
3. Create the CampaignResponse Table

▶ Task 1: Create the Competitor Table

1. In the Lab Exercise 3.sql file, write and execute a query to create the **Competitor** table that you designed in Exercise 1 in the **DirectMarketing** schema.
2. In Object Explorer, verify that the new table exists.

▶ Task 2: Create the TVAdvertisement Table

1. In the Lab Exercise 3.sql file, write and execute a query to create the **TVAdvertisement** table that you designed in Exercise 1 in the **DirectMarketing** schema.
2. Refresh Object Explorer and verify that the new table exists.

▶ Task 3: Create the CampaignResponse Table

1. In the Lab Exercise 3.sql file, write and execute a query to create the **CampaignResponse** table that you designed in Exercise 1 in the **DirectMarketing** schema.
2. Refresh Object Explorer and verify that the new table exists.
3. Review the **Computed text** property of the **ResponseProfit** column.
4. Close SQL Server Management Studio without saving changes.

Results: After completing this exercise you will have created the Competitor, TVAdvertisement, and the CampaignResponse tables. You will have created table columns with the appropriate NULL or NOT NULL settings, and primary keys.

Question: When should a column be declared as nullable?

Lab answer Key 2: Designing and Implementing Tables

Exercise 1: Designing Tables

▶ Task 1: Prepare the Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab02\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Review the Design

1. In File Explorer, browse to the **D:\Labfiles\Lab02\Starter** folder, and then double-click **Schema Design for Marketing Development Tables.docx**.
2. Review the proposed structure for the three tables.

▶ Task 3: Improve the Design

1. In the Schema Design document, add Yes or No to the **Allow Nulls?** column for each table, depending whether you think that column should allow nulls.
2. On the **File** menu, click **Save**.
3. On the **File** menu, click **Close**.
4. On the **File** menu, click **Open**.
5. In the **Open** pane, click **Browse**.
6. In the **Open** dialog box, browse to the **D:\Labfiles\Lab02\Solution** folder, click **Schema Design for Marketing Development Tables.docx**, and then click **Open**.
7. Review the suggested nullability for the columns in the three tables.
8. Close WordPad without saving changes.

Results: After completing this exercise, you will have an improved schema and table design.

Exercise 2: Creating Schemas

▶ Task 1: Create a Schema

1. On the taskbar, click **Microsoft SQL Server Management Studio**.
2. In the **Connect to Server** dialog box, in the **Server name** box, type **MIA-SQL**, in the **Authentication** list, click **Windows Authentication**, and then click **Connect**.
3. On the **File** menu, point to **Open**, and then click **Project/Solution**.
4. In the **Open Project** dialog box, navigate to **D:\Labfiles\Lab02\Starter\Project**, and then double-click **Project.ssmssln**.
5. In Solution Explorer, double-click **Lab Exercise 2.sql**.
6. Select the **USE TSQL** statement, and then click **Execute**.
7. Under the comment that starts **Task 1**, type the following query, and then click **Execute**:

```
CREATE SCHEMA DirectMarketing
AUTHORIZATION dbo;
GO
```

8. Leave SQL Server Management Studio open for the next exercise.

Results: After completing this exercise, you will have a new schema in the database.

Exercise 3: Creating Tables

▶ Task 1: Create the Competitor Table

1. In Solution Explorer, double-click the query **Lab Exercise 3.sql**.
2. Select the **USE TSQL** statement, and then click **Execute**.
3. Under the comment that starts **Task 1**, type the following query, select the query, and then click **Execute**:

```
CREATE TABLE DirectMarketing.Competitor
(
CompetitorCode nvarchar(6) NOT NULL,
Name varchar(30) NOT NULL,
[Address] varchar(max) NULL,
Date_Entered varchar(10) NULL,
Strength_of_competition varchar(8) NULL,
Comments varchar(max) NULL
);
GO
```

4. In Object Explorer, expand **Databases**, expand **TSQL**, expand **Tables**, and verify that the **DirectMarketing.Competitor** table exists.

▶ Task 2: Create the TVAdvertisement Table

1. Under the comment that starts **Task 2**, type the following query, select the query, and then click **Execute**:

```
CREATE TABLE DirectMarketing.TVAdvertisement
(
TV_Station nvarchar(15) NOT NULL,
City nvarchar(25) NULL,
CostPerAdvertisement float NULL,
TotalCostOfAllAdvertisements float NULL,
NumberOfAdvertisements varchar(4) NULL,
Date_Of_Advertisement_1 varchar(12) NULL,
Time_Of_Advertisement_1 int NULL,
Date_Of_Advertisement_2  varchar(12) NULL,
Time_Of_Advertisement_2 int NULL,
Date_Of_Advertisement_3  varchar(12) NULL,
Time_Of_Advertisement_3 int NULL,
Date_Of_Advertisement_4  varchar(12) NULL,
Time_Of_Advertisement_4 int NULL,
Date_Of_Advertisement_5  varchar(12) NULL,
Time_Of_Advertisement_5  int NULL
);
GO
```

2. In Object Explorer, under **TSQL**, right-click **Tables**, and then click **Refresh**.
3. Verify that the **DirectMarketing.TVAdvertisement** table exists.

▶ Task 3: Create the CampaignResponse Table

1. Under the comment that starts **Task 3**, type the following query, select the query, and then click **Execute**:

```
CREATE TABLE DirectMarketing.CampaignResponse
(
ResponseOccurredWhen datetime,
RelevantProspect int,
RespondedHow varchar(8),
ChargeFromReferrer decimal(8,2),
RevenueFromResponse decimal(8,2),
ResponseProfit AS (RevenueFromResponse - ChargeFromReferrer) PERSISTED
);
GO
```

2. In Object Explorer, under **TSQL**, right-click **Tables**, and then click **Refresh**.
3. Verify that the **DirectMarketing.CampaignResponse** table exists.
4. Expand **DirectMarketing.CampaignResponse**, and then expand **Columns**. Note that the **ResponseProfit** column is defined as a **Computed** column.
5. Double-click **ResponseProfit** and in the **Column Properties - ResponseProfit** dialog box, review the **Computed text** property, and then click **Cancel**.
6. Close SQL Server Management Studio without saving changes.

Results: After completing this exercise you will have created the Competitor, TVAdvertisement, and the CampaignResponse tables. You will have created table columns with the appropriate NULL or NOT NULL settings, and primary keys.

Lab 3: Ensuring Data Integrity Through Constraints

Scenario

A table named Opportunity has recently been added to the DirectMarketing schema within the Adventureworks database, but it has no constraints in place. In this lab, you will implement the required constraints to ensure data integrity and, ifyou have time, test that constraints work as specified.

Column Name	Data Type	Required	Validation Rule
OpportunityID	Int	Yes	Part of the Primary key
ProspectID	Int	Yes	Part of the key—also, prospect must exist
DateRaised	datetime	Yes	Must be today's date
Likelihood	Bit	Yes	
Rating	char(1)	Yes	
EstimatedClosingDate	date	Yes	
EstimatedRevenue	decimal(10,2)	Yes	

Objectives

After completing this lab, you will be able to:

- Use the ALTER TABLE statement to adjust the constraints on existing tables.
- Create and test a DEFAULT constraint.
- Create and test a CHECK constraint.
- Create and test a UNIQUE constraint.
- Create and test a PRIMARY KEY constraint.
- Create and test a Referential Integrity FOREIGN KEY constraint.
- Create and test a CASCADING REFERENTIAL INTEGRITY constraint for a FOREIGN KEY and a PRIMARY KEY.

Virtual machine: **20762C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Add Constraints

Scenario

You have been given the design for a table called DirectMarketing.Opportunity. You must alter the table with the appropriate constraints, based upon the provided specifications.

The main tasks for this exercise are as follows:

- Review the supporting documentation.
- Alter the DirectMarketing.Opportunity table.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Review Supporting Documentation
3. Alter the Direct Marketing Table

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. On the taskbar, click **File Explorer**.
3. In File Explorer, navigate to the **D:\Labfiles\Lab04\Starter** folder, right-click the **Setup.cmd** file, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Review Supporting Documentation

- Review the table design requirements that were supplied in the scenario.

▶ Task 3: Alter the Direct Marketing Table

1. Work through the list of requirements and alter the table to make the columns required, based on the requirements.
2. Work through the list of requirements and alter the table to make columns the primary key, based on the requirements.
3. Work through the list of requirements and alter the table to make columns foreign keys, based on the requirements.
4. Work through the list of requirements and alter the table to add DEFAULT constraints to columns, based on the requirements.

Exercise 2: Test the Constraints

Scenario

You should now test each of the constraints that you designed to ensure that they work as expected.

The main tasks for this exercise are as follows:

- Test the default values and data types.
- Test the primary key.
- Test the foreign key reference on **ProspectID**.

The main tasks for this exercise are as follows:

1. Test the Data Types and Default Constraints
2. Test the Primary Key
3. Test to Ensure the Foreign Key is Working as Expected

▶ Task 1: Test the Data Types and Default Constraints

- Create a new query for the solution called **ConstraintTesting.sql**. Use this new connection to Adventureworks to insert a row into the opportunity table using the following values, which are organized by the columns as found within the table: [**1,1,8,'A','12/12/2013',123000.00**]. Note that we do not specify the column DateRaised in column list (nor a value in the value list) since we want that column's default to be applied.

▶ Task 2: Test the Primary Key

- Try to add the same row again to confirm that the primary key constraint is working to ensure entity integrity—only unique rows can be added to the table

▶ Task 3: Test to Ensure the Foreign Key is Working as Expected

- Try to add some data for a prospect that does not exist, to confirm that the foreign key constraint is working, to ensure relational integrity. Only nonunique rows are to be added to the table for foreign key values that are uniquely available in the prospects table.

Results: After completing this exercise, you should have successfully tested your constraints.

Lab Answer Key 3: Ensuring Data Integrity Through Constraints

Exercise 1: Add Constraints

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. On the taskbar, click **File Explorer**.
3. In File Explorer, navigate to the **D:\Labfiles\Lab04\Starter** folder, right-click the **Setup.cmd** file, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Review Supporting Documentation

- Review the table design requirements that were supplied in the scenario

▶ Task 3: Alter the Direct Marketing Table

1. In File Explorer, navigate to the directory **D:\Labfiles\Lab04\Starter**, double-click on the starter project, which is named **Starter.ssmssqlproj**.
2. In SQL Server Management Studio, in Solution Explorer, in the **Queries** folder, double-click **AlterTable.sql**.
3. Highlight the code below **Step 1: Make sure your database scope is set to Adventureworks**, and click **Execute**.
4. Highlight the code below **Step 2: Explore the metadata for DirectMarketing.Opportunity**, and click **Execute**.
5. Highlight the code below **Step 3: Alter the table to meet the requirements**, and click **Execute**.
6. Under the **Step 4: Add a composite primary key to the table**, type the following code:

```
ALTER TABLE DirectMarketing.Opportunity
ADD CONSTRAINT PK_Opportunity PRIMARY KEY CLUSTERED (OpportunityID, ProspectID);
GO
```

7. Highlight the code and click **Execute**.
8. Under the **Step 5: Add a foreign key to the table, linking it to the Prospect table primary key**, type the following code:

```
ALTER TABLE DirectMarketing.Opportunity
ADD CONSTRAINT FK_OpportunityProspect
FOREIGN KEY (ProspectID) REFERENCES DirectMarketing.Prospect(ProspectID);
GO
```

9. Highlight the code and click **Execute**.

10. Under the **Step 6: Add a default constraint that will set the DateRaised column to the current system data and time**, type the following code:

```
ALTER TABLE DirectMarketing.Opportunity
ADD CONSTRAINT dfDateRaised
DEFAULT (SYSDATETIME()) FOR DateRaised;
GO
```

11. Highlight the code and click **Execute**.
12. Under the **Step 7: Explore the metatdata for DirectMarketing.Opportunity**, investigate the changes to the metadata by typing the following code:

```
sp_help 'DirectMarketing.Opportunity'
```

13. Highlight the code and click **Execute**.

Exercise 2: Test the Constraints

▶ Task 1: Test the Data Types and Default Constraints

1. In Solution Explorer, right-click the **Queries** folder, and then click **New Query**.
2. Right-click the new file, click **Rename**, type **ConstraintTesting.sql**, and then press Enter.
3. In the query pane, type the following query:

```
-- Step 1: Insert a new row to test the data types and default behavior within the
DirectMarketing.Opportunity table
INSERT INTO DirectMarketing.Opportunity (OpportunityID,ProspectID,
Likelihood,Rating,EstimatedClosingDate, EstimatedRevenue)
VALUES (1,1,8,'A','12/12/2013',123000.00);
SELECT * FROM DirectMarketing.Opportunity;
GO
```

4. Highlight the code and click **Execute**.

Note: This query should execute without errors.

▶ Task 2: Test the Primary Key

1. In the query pane, under the existing code, type the following query:

```
-- Step 2: Try to add the same data again to test that the primary key prevents this
action
INSERT INTO DirectMarketing.Opportunity (OpportunityID,ProspectID,
Likelihood,Rating,EstimatedClosingDate, EstimatedRevenue)
VALUES (1,1,8,'A','12/12/2013',123000.00);
GO
```

2. Highlight the code and click **Execute**.

Note: This query should fail due to the PRIMARY KEY constraint.

▶ Task 3: Test to Ensure the Foreign Key is Working as Expected

1. In the query pane, under the existing code, type the following query:

```
-- Step 3: Try to add some data for a prospect that does not exist
INSERT INTO DirectMarketing.Opportunity (OpportunityID,ProspectID,
Likelihood,Rating,EstimatedClosingDate, EstimatedRevenue)
VALUES (2,10,8,'A','12/12/2013',123000.00);
GO
```

2. Highlight the code and click **Execute**.

Note: This query should fail due to the FOREIGN KEY constraint.

3. Close SQL Server Management Studio without saving any changes.

Results: After completing this exercise, you should have successfully tested your constraints.

Lab 4: Implementing Indexes

Scenario

One of the most important decisions when designing a table is to choose an appropriate table structure. In this lab, you will choose an appropriate structure for some new tables required for the relationship management system.

Objectives

After completing this lab, you will be able to:

- Create a table without any indexes.
- Create a table with a clustered index.
- Add a nonclustered key in the form of a covering index.

Virtual machine: **20762C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Creating a Heap

Scenario

The design documentation requires you to create tables to store sales related data. You will create these two tables to support the requirement of the sales department.

The supporting documentation for this exercise is located in **D:\Labfiles\Lab05\Starter\Supporting Documentation.docx**.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Review the Documentation
3. Create the Tables

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab05\Starter** folder as Administrator.

▶ Task 2: Review the Documentation

- Review the requirements in **Supporting Documentation.docx** in the **D:\Labfiles\Lab05\Starter** folder and decide how you are going to meet them.

▶ Task 3: Create the Tables

1. Create a table based on the supporting documentation for Table 1: Sales.MediaOutlet.
2. Create a table based on the supporting documentation for Table 2: Sales.PrintMediaPlacement.

Results: After completing this exercise, you will have created two new tables in the AdventureWorks database.

Exercise 2: Creating a Clustered Index

Scenario

The sales department has started to use the new tables, and are finding that, when trying to query the data, the performance is unacceptable. They have asked you to make any database changes you can to improve performance.

The main tasks for this exercise are as follows:

1. Add a Clustered Index to Sales.MediaOutlet
2. Add a Clustered Index to Sales.PrintMediaPlacement

▶ Task 1: Add a Clustered Index to Sales.MediaOutlet

1. Consider which column is best suited to an index.
2. Using Transact-SQL statements, add a clustered index to that column on the **Sales.MediaOutlet** table.

   Consider implementing the index by creating a unique constraint.
3. Use Object Explorer to check that the index was created successfully.

▶ Task 2: Add a Clustered Index to Sales.PrintMediaPlacement

1. Consider which column is best suited to an index.
2. Using Transact-SQL statements, add a clustered index to that column on the **Sales.PrintMediaPlacement** table.
3. Use Object Explorer to check that the index was created successfully.

Results: After completing this exercise, you will have created clustered indexes on the new tables.

Exercise 3: Creating a Covering Index

Scenario

The sales team has found that the performance improvements that you have made are not working for one specific query. You have been tasked with adding additional performance improvements to handle this query.

The main tasks for this exercise are as follows:

1. Add Some Test Data
2. Run the Poor Performing Query
3. Create a Covering Index
4. Check the Performance of the Sales Query

▶ Task 1: Add Some Test Data

- Run the Transact-SQL in **D:\Labfiles\Lab05\Starter\InsertDummyData.sql** to insert test data into the two tables.

▶ Task 2: Run the Poor Performing Query

1. Switch on Include Actual Execution Plan.
2. Run the Transact-SQL in **D:\Labfiles\Lab05\Starter\SalesQuery.sql**.
3. Examine the Execution Plan.
4. Note the missing index warning in SQL Server Management Studio.

▶ Task 3: Create a Covering Index

1. On the **Execution Plan** tab, right-click the green **Missing Index** text and click **Missing Index Details**.
2. Use the generated Transact-SQL to create the missing covering index.
3. Use Object Explorer to check that the index was created successfully.

▶ Task 4: Check the Performance of the Sales Query

1. Rerun the sales query.
2. Check the Execution Plan and ensure the database engine is using the new **NCI_PrintMediaPlacement** index.
3. Close SQL Server Management Studio without saving any changes.

Results: After completing this exercise, you will have created a covering index suggested by SQL Server Management Studio.

Lab Answer Key 4: Implementing Indexes

Exercise 1: Creating a Heap

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab05\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Review the Documentation

1. In File Explorer, navigate to the **D:\Labfiles\Lab05\Starter** folder and then double-click **Supporting Documentation.docx**.
2. Review the requirements in the supporting documentation for the tables, and decide how you are going to meet them.

▶ Task 3: Create the Tables

1. On the taskbar, click **Microsoft SQL Server Management Studio**.
2. In the **Connect to Server** dialog box, in the **Server name** box, type **MIA-SQL**, in the **Authentication** list, click **Windows Authentication**, and then click **Connect**.
3. In Object Explorer, expand **Databases**, right-click **AdventureWorks**, and then click **New Query**.
4. Type the following query in the query pane:

```
CREATE TABLE Sales.MediaOutlet (
MediaOutletID INT NOT NULL,
MediaOutletName NVARCHAR(40),
PrimaryContact NVARCHAR (50),
City NVARCHAR (50)
);
```

5. On the toolbar, click **Execute**.
6. In Object Explorer, right-click **AdventureWorks**, and in the context menu click **New Query**.
7. Type the following query in the query pane:

```
CREATE TABLE Sales.PrintMediaPlacement ( PrintMediaPlacementID INT NOT NULL,
MediaOutletID INT,
PlacementDate DATETIME,
PublicationDate DATETIME,
RelatedProductID INT,
PlacementCost DECIMAL(18,2)
);
```

8. On the toolbar, click **Execute**.
9. In Object Explorer, expand **AdventureWorks**, and then expand **Tables**. Note that your two new tables are included in the list. If no new tables appear, in Object Explorer, click **Refresh**.
10. Expand **Sales.MediaOutlet**, expand **Indexes**, and note that there are no indexes on the table.

11. Repeat Step 10 for the **Sales.PrintMediaPlacement** table.
12. Leave SQL Server Management Studio open for the next exercise.

Results: After completing this exercise, you will have created two new tables in the AdventureWorks database.

Exercise 2: Creating a Clustered Index

▶ Task 1: Add a Clustered Index to Sales.MediaOutlet

1. In Object Explorer, right-click **AdventureWorks**, and then click **New Query**.
2. Type the following query in the query pane:

```
ALTER TABLE Sales.MediaOutlet ADD CONSTRAINT IX_MediaOutlet UNIQUE CLUSTERED (
MediaOutletID
);
```

3. On the toolbar, click **Execute**.
4. In Object Explorer, click **Sales.MediaOutlet**, and then on the toolbar, click **Refresh**.
5. Expand **Indexes** and note that the table has a clustered index named **IX_MediaOutlet (Clustered)**.
6. Expand **Keys** and note that the table has a key named **IX_MediaOutlet**.

▶ Task 2: Add a Clustered Index to Sales.PrintMediaPlacement

1. In Object Explorer, right-click **AdventureWorks**, and then click **New Query**.
2. Type the following query in the query pane:

```
CREATE UNIQUE CLUSTERED INDEX CIX_PrintMediaPlacement ON Sales.PrintMediaPlacement (
PrintMediaPlacementID ASC
);
```

3. On the toolbar, click **Execute**.
4. In Object Explorer, click **Sales.PrintMediaPlacement**, and then on the toolbar, click **Refresh**.
5. Expand **Indexes** and note that the table has a clustered index named **CIX_PrintMediaPlacement (Clustered)**.
6. Leave SQL Server Management Studio open for the next exercise.

Results: After completing this exercise, you will have created clustered indexes on the new tables.

Exercise 3: Creating a Covering Index

▶ Task 1: Add Some Test Data

1. On the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab05\Starter** folder, click **InsertDummyData.sql**, and then click **Open**.
3. On the toolbar, click **Execute.**

▶ Task 2: Run the Poor Performing Query

1. On the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab05\Starter** folder, click **SalesQuery.sql**, and then click **Open**.
3. On the **Query** menu, click **Include Actual Execution Plan**.
4. On the toolbar, click **Execute**.
5. On the **Execution Plan** tab, note the missing index warning.

▶ Task 3: Create a Covering Index

1. On the **Execution Plan** tab, move the mouse over the green **Missing Index** text.
2. Right-click the green **Missing Index** text, and then click **Missing Index Details**.
3. In the query pane, delete the **/*** on line 6 and then delete the ***/** on line 13.
4. Modify the query so that it reads as follows:

```
USE [AdventureWorks]
GO
CREATE NONCLUSTERED INDEX NCI_PrintMediaPlacement
ON [Sales].[PrintMediaPlacement] ([PublicationDate],[PlacementCost])
INCLUDE ([PrintMediaPlacementID],[MediaOutletID],[PlacementDate],[RelatedProductID])
GO
```

5. On the toolbar, click **Execute**.
6. In Object Explorer, click **Sales.PrintMediaPlacement**, and then on the toolbar, click **Refresh**.
7. Expand **Indexes** and note that there is a nonclustered index named **NCI_PrintMediaPlacement (Non-Unique, Non-Clustered)**.

▶ Task 4: Check the Performance of the Sales Query

1. Switch to the **SalesQuery.sql** query file, and on the toolbar, click **Execute**.
2. On the **Execution Plan** tab, note that SQL Server Management Studio no longer warns of a missing index.
3. Note that the new **NCI_PrintMediaPlacement** index is being used.
4. Close SQL Server Management Studio without saving any changes.

Results: After completing this exercise, you will have created a covering index suggested by SQL Server Management Studio.

Lab 5: Optimizing Indexes

Scenario

You have been hired by the IT Director of the Adventure Works Bicycle Company to work with their DBA to improve the use of indexes in the database. You want to show the DBA how to use Query Store to improve query performance and identify missing indexes. You will also highlight the importance of having a clustered index on each table.

Objectives

In this lab, you will practice:

- Using Query Store to monitor queries and identify missing indexes.
- Compare a heap against a table with a clustered index.

Virtual machine: **20762C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Using Query Store

Scenario

You are the DBA for the Adventure Works Bicycle Company. You have been working with a consultant to implement the features in Query Store, and now want to simulate a typical query load.

The main tasks for this exercise are as follows:

1. Use Query Store to Monitor Query Performance

▶ Task 1: Use Query Store to Monitor Query Performance

1. Log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab06\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In SSMS, connect to the **MIA-SQL** database engine instance using Windows authentication.
4. Open **QueryStore_Lab1.sql**.
5. Make **AdventureWorks2016** the current database.
6. Execute the code to create an indexed view.
7. Select the code under the comment **Clear the Query Store**, and then click **Execute**.
8. Select the code under the comment **Run a select query six times**, and then click **Execute**.
9. Repeat five times, waiting a few seconds each time.
10. Select the code to **Update the statistics with fake figures**, and then click **Execute**.
11. Repeat Step 10 twice.
12. In **Query Store**, double-click **Top Resource Consuming Queries**.
13. Examine the different views of the Top 25 Resources report and force the original query plan to be used.
14. Switch to the **QueryStore_Lab1.sql** tab and repeat Step 10 three times.

15. Switch to the **Top Resource Consuming Queries** tab to identify which query plans used a clustered index seek and which ones used a clustered index scan.
16. Keep SSMS open for the next lab exercise.

Results: After completing this lab exercise you will have used Query Store to monitor query performance, and used it to force a particular execution plan to be used.

Exercise 2: Heaps and Clustered Indexes

Scenario

You are the DBA for the Adventure Works Bicycle Company. You have had complaints that a number of queries have been running slowly. When you take a closer look, you realize that a number of tables have been created without a clustered index. Before adding a clustered index, you decide to run some tests to find out what difference a clustered index makes to query performance.

The main tasks for this exercise are as follows:

1. Compare a Heap with a Clustered Index

▶ Task 1: Compare a Heap with a Clustered Index

1. Open **ClusterVsHeap_lab.sql**, and run each part of the script in turn.
2. Make **AdventureWorks2016** the current database.
3. Run the script to create a table as a heap.
4. Run the script to create a table with a clustered index.
5. Run the script to SET STATISTICS ON. Run each set of select statements on both the heap, and the clustered index.
6. Open **HeapVsClustered_Timings.docx**, and use the document to note the CPU times for each.
7. Run the script to select from each table.
8. Run the script to select from each table with the ORDER BY clause.
9. Run the script to select from each table with the WHERE clause.
10. Run the script to select from each table with both the WHERE clause and ORDER BY clause.
11. Run the script to insert data into each table.
12. Compare your results with the timings in the Solution folder.
13. If you have time, run the select statements again and **Include Live Query Statistics**.
14. Close SQL Server Management Studio, without saving any changes.
15. Close Wordpad.

Results: After completing this lab exercise, you will:

Understand the effect of adding a clustered index to a table.

Understand the performance difference between a clustered index and a heap.

Lab Answer Key 5: Optimizing Indexes

Exercise 1: Using Query Store

▶ Task 1: Use Query Store to Monitor Query Performance

1. Ensure that the **20762C-MIA-DC** and 20**762C-MIA-SQL** virtual machines are running.
2. Log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. In the **D:\Labfiles\Lab06\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and wait for the script to finish.
5. Start SSMS and connect to the **MIA-SQL** database engine instance using Windows authentication.
6. On the **File** menu, point to **Open**, and then click **File**.
7. In the **Open File** dialog box, navigate to **D:\Labfiles\Lab06\Starter**, click **QueryStore_Lab1.sql**, and then click **Open**.
8. Select the code under the comment **Use the AdventureWorks2016 database**, and then click **Execute**.
9. Select the code under the comment **Create indexed view**, and then click **Execute**.
10. Select the code under the comment **Clear the Query Store**, and then click **Execute**.
11. Select the code under the comment **Run a select query six times**, and then click **Execute**.
12. Repeat the last step another five times, waiting a few seconds between each execution.
13. Select the code under the comment **Update the statistics with fake figures**, and then click **Execute**.
14. Repeat Step 11 twice, waiting a few seconds between each execution.
15. In Object Explorer, expand the **Databases** node, expand **AdventureWorks2016**, expand **Query Store**, and double-click **Top Resource Consuming Queries**.
16. Examine the different views of the Top 25 Resources report.
17. Locate the original query plan, and click **Force Plan**.
18. In the **Confirmation** dialog box, click **Yes**.
19. Switch to the **QueryStore_Lab1.sql** tab.
20. Repeat Step 11 three times, waiting a few seconds between each execution.
21. Switch to the **Top Resource Consuming Queries** tab.
22. Using the different views of the report, identify which query plans used a clustered index seek and which ones used a clustered index scan.
23. Keep SSMS open for the next lab exercise.

Results: After completing this lab exercise you will have used Query Store to monitor query performance, and used it to force a particular execution plan to be used.

Exercise 2: Heaps and Clustered Indexes

▶ Task 1: Compare a Heap with a Clustered Index

1. On the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, navigate to **D:\Labfiles\Lab06\Starter**, click **ClusterVsHeap_lab.sql**, and then click **Open**.
3. Select the code under the comment **ClusterVsHeap_lab**, and then click **Execute** to make **AdventureWorks2016** the current database.
4. Select the code under the comment **Create a heap and a clustered index**, and then click **Execute**.
5. Select the code under the comment **SET STATISTICS ON**, and then click **Execute**. The script includes a number of different select statements that will be run on both the heap, and the clustered index.
6. Using file explorer, navigate to **D:\Labfiles\Lab06\Starter** and open **HeapVsClustered_Timings.docx**. Use the document to note the CPU time whilst running the code in the following steps.
7. In SSMS, select the code under the comment **SELECT**, and then click **Execute**.
8. On the **Messages** tab, note the CPU time.
9. Select the code under the comment **SELECT ORDER BY**, and then click **Execute**.
10. On the **Messages** tab, note the CPU time.
11. Select the code under the comment **SELECT WHERE**, and then click **Execute**.
12. On the **Messages** tab, note the CPU time.
13. Select the code under the comment **SELECT WHERE ORDER BY**, and then click **Execute**.
14. On the **Messages** tab, note the CPU time.
15. Select the code under the comment **INSERT**, and then click **Execute**.
16. On the **Messages** tab, note the CPU time.
17. Select the code under the comment **SELECT ORDER BY**, and then click **Execute**.
18. On the **Messages** tab, note the CPU time.
19. Compare your results with the timings in the **D:\Labfiles\Lab06\Solution\HeapVsClustered_Solution.docx** file.
20. If you have time, run the select statements again with Live Query Statistics. Click the **Include Live Query Statistics** button on the toolbar, and run each query again.
21. Close SQL Server Management Studio, without saving any changes.
22. Close Wordpad.

Results: After completing this lab exercise, you will:

Understand the effect of adding a clustered index to a table.

Understand the performance difference between a clustered index and a heap.

Lab 6: Designing and Implementing Views

Scenario

A new web-based stock promotion is being tested at the Adventure Works Bicycle Company. Your manager is worried that providing access from the web-based system directly to the database tables will be insecure, so has asked you to design some views for the web-based system.

The Sales department has also asked you to create a view that enables a temporary worker to enter new customer data without viewing credit card, email address, or phone number information.

Objectives

After completing this lab, you will be able to:

- Create standard views.
- Create updateable views.

Virtual machine: **20762C-MIA-SQL**

User name: **AdventureWorks\Student**

Password: **Pa55w.rd**

Exercise 1: Creating Standard Views

Scenario

The web-based stock promotion requires two new views: OnlineProducts and Available Models. The documentation for each view is shown in the following tables:

View 1: OnlineProducts

View Column	Table Column
ProductID	Production.Product,ProductID
Name	Production.Product,Name
Product Number	Production.Product,ProductNumber
Color	Production.Product.Color. If NULL, return 'N/A'
Availability	Production.Product.DaysToManufacture. If 0 returns 'In stock', If 1 returns 'Overnight'. If 2 return '2 to 3 days delivery'. Otherwise, return 'Call us for a quote'.
Size	Production.Product.Size
Unit of Measure	Production.Product.SizeUnitMeasureCode
Price	Production.Product.ListPrice
Weight	Production.Product.Weight

This view is based on the Production.Product table. Products should be displayed only if the product is on sale, which can be determined using the SellStartDate and SellEndDate columns.

View 2: Available Models

View Column	Table Column
Product ID	Production.Product.ProductID
Product Name	Production.Product.Name
Product Model ID	Production.ProductModel.ProductModelID
Product Model	Production.ProductMode.Name

This view is based on two tables: Production.Product and Production.ProductModel. Products should be displayed only if the product is on sale, which can be determined using the SellStartDate and SellEndDate columns.

The main tasks for this exercise are as follows:

1. Prepare the Environment
2. Design and Implement the Views
3. Test the Views

▶ Task 1: Prepare the Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab08\Starter** folder as Administrator.

▶ Task 2: Design and Implement the Views

1. Review the documentation for the new views.
2. Using SSMS, connect to MIA-SQL using Windows Authentication.
3. Open a new query window.
4. Write and execute scripts to create the new views.

▶ Task 3: Test the Views

- Query both views to ensure that they return the data required in the original documentation.

Results: After completing this exercise, you will have two new views in the AdventureWorks database.

Exercise 2: Creating an Updateable View

Scenario

The Sales department has asked you to create an updateable view based on the Sales.CustomerPII table, enabling a temporary worker to enter a batch of new customers while keeping the credit card, email and phone number information secure.

The view must contain three columns from the Sales.CustomerPII table: CustomerID, FirstName and LastName. You must be able to update the view with new customers.

View Columns	Table Columns
CustomerID	Sales.CustomerPII.CustomerID
FirstName	Sales.CustomerPII.FirstName
LastName	Sales.CustomerPII.LastName

The main tasks for this exercise are as follows:

1. Design and Implement the Updateable View
2. Test the Updateable View

▶ Task 1: Design and Implement the Updateable View

1. Review the requirements for the updateable view.
2. Write and execute a script to create the new view.

▶ Task 2: Test the Updateable View

1. Write and execute a SELECT query to check that the view returns the correct columns. Order the result set by CustomerID.
2. Write and execute an INSERT statement to add a new record to the view.
3. Check that the new record appears in the view results.
4. Close SSMS without saving any changes.

Results: After completing this exercise, you will have a new updateable view in the database.

Lab Answer Key 6: Designing and Implementing Views

Exercise 1: Creating Standard Views

▶ Task 1: Prepare the Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab08\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Design and Implement the Views

1. Review the documentation for the views.
2. On the taskbar, click **Microsoft SQL Server Management Studio**.
3. In the **Connect to Server** dialog box, in the **Server name** box, type **MIA-SQL**, in the **Authentication** list, click **Windows Authentication**, and then click **Connect**.
4. On the toolbar, click **New Query**.
5. In the new query pane, type the following code to create the **Production.OnlineProducts** view:

```
USE AdventureWorks2016;
GO
CREATE VIEW
Production.OnlineProducts
AS
SELECT p.ProductID, p.Name, p.ProductNumber AS [Product Number], COALESCE(p.Color,
'N/A') AS Color,
CASE p.DaysToManufacture
WHEN 0 THEN 'In stock'
WHEN 1 THEN 'Overnight'
WHEN 2 THEN '2 to 3 days delivery'
ELSE 'Call us for a quote'
END AS Availability,
p.Size, p.SizeUnitMeasureCode AS [Unit of Measure], p.ListPrice AS Price, p.Weight
FROM Production.Product AS p
WHERE p.SellEndDate IS NULL AND p.SellStartDate IS NOT NULL;
GO
```

6. Below the query that you have just typed, type the following code to create the **Production.AvailableModels** view:

```
USE AdventureWorks2016;
GO
CREATE VIEW
Production.AvailableModels
AS
SELECT p.ProductID AS [Product ID], p.Name, pm.ProductModelID AS [Product Model ID],
pm.Name as [Product Model]
FROM Production.Product AS p
INNER JOIN Production.ProductModel AS pm
ON p.ProductModelID = pm.ProductModelID
WHERE p.SellEndDate IS NULL
AND p.SellStartDate IS NOT NULL;
```

```
GO
```

7. On the toolbar, click **Execute** to create the views.

▶ Task 3: Test the Views

1. On the **File** menu, point to **New**, and then click **Query with Current Connection**.
2. In the new query pane, type the following code to test the new views:

```
USE AdventureWorks2016;
GO
SELECT * FROM Production.OnlineProducts;
GO
SELECT * FROM Production.AvailableModels;
GO
```

3. On the toolbar, click **Execute**.
4. Check that each view is returning the correct columns, check that the column headings are correct, and check that only products that are available to be sold are listed.
5. Leave SSMS open for use in the next exercise.

Results: After completing this exercise, you will have two new views in the AdventureWorks database.

Exercise 2: Creating an Updateable View

▶ Task 1: Design and Implement the Updateable View

1. Review the requirements for the updateable view.
2. In SSMS, on the **File** menu, point to **New**, and then click **Query with Current Connection**.
3. In the new query pane, type the following code to create the updateable **Sales.NewCustomer** view:

```
USE AdventureWorks2016;
GO
CREATE VIEW Sales.NewCustomer
AS
SELECT CustomerID, FirstName, LastName
FROM Sales.CustomerPII;
GO
```

4. On the toolbar, click **Execute** to create the view.

▶ Task 2: Test the Updateable View

1. On the **File** menu, point to **New**, and then click **Query with Current Connection**.
2. In the new query pane, type the following code to test that the new view returns the correct data:

```
USE AdventureWorks2016;
GO
SELECT * FROM Sales.NewCustomer
ORDER BY CustomerID
```

3. On the toolbar, click **Execute**.

4. Check that the view is returning the correct columns.
5. Above the query that you have just typed, type the following code to test that the new view is updateable:

```
USE AdventureWorks2016;
GO
INSERT INTO Sales.NewCustomer
VALUES
(10001,'Ed', 'Kish'),
(10002, 'Kermit', 'Albritton');
GO
```

6. On the toolbar, click **Execute**.
7. Check that the new rows now appear in the results set from the view.
8. Close SSMS without saving any changes.

Results: After completing this exercise, you will have a new updateable view in the database.

Lab 7: Designing and Implementing Stored Procedures

Scenario

You need to create a set of stored procedures to support a new reporting application. The procedures will be created within a new Reports schema.

Objectives

After completing this lab, you will be able to:

Create a stored procedure.

- Change the execution context of a stored procedure.
- Create a parameterized stored procedure.

Virtual machine: **20762C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Create Stored Procedures

Scenario

In this exercise, you will create two stored procedures in the MarketDev database to support one of the new reports.

Supporting Documentation

Stored Procedure:	**Reports.GetProductColors**
Input Parameters:	None
Output Parameters:	None
Output Columns:	Color (from Marketing.Product)
Notes:	Colors should not be returned more than once in the output. NULL values should not be returned.
Stored Procedure:	Reports.GetProductsAndModels
Input Parameters:	None
Output Parameters:	None
Output Columns:	ProductID, ProductName, ProductNumber, SellStartDate, SellEndDate and Color (from Marketing.Product), ProductModelID (from Marketing.ProductModel), EnglishDescription, FrenchDescription, ChineseDescription.
Output Order:	ProductID, ProductModelID.
Notes:	For descriptions, return the Description column from the Marketing.ProductDescription table for the appropriate language. The LanguageID for English is “en”, for French is “fr” and for Chinese is “zh-cht”. If no specific language description is available, return the invariant

Stored Procedure:	Reports.GetProductColors
	language description if it is present. The LanguageID for the invariant language is a blank string ''. Where neither the specific language nor invariant language descriptions exist, return the ProductName instead.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Review the Reports.GetProductColors Stored Procedure Specification
3. Design, Create and Test the Reports.GetProductColors Stored Procedure
4. Review the Reports.GetProductsAndModels Stored Procedure Specification
5. Design, Create and Test the Reports.GetProductsAndModels Stored Procedure

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab09\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and wait for the script to finish.

▶ Task 2: Review the Reports.GetProductColors Stored Procedure Specification

- Review the design requirements in the Exercise Scenario for Marketing.GetProductColors.

▶ Task 3: Design, Create and Test the Reports.GetProductColors Stored Procedure

- Design and implement the stored procedure in accordance with the design specifications.

▶ Task 4: Review the Reports.GetProductsAndModels Stored Procedure Specification

- Review the supplied design requirements in the supporting documentation in the Exercise Scenario for Reports.GetProductsAndModels.

▶ Task 5: Design, Create and Test the Reports.GetProductsAndModels Stored Procedure

- Design and implement the stored procedure in accordance with the design specifications.

Results: After completing this lab, you will have created and tested two stored procedures, Reports.GetProductColors and Reports.GetProductsAndModels.

Exercise 2: Create Parameterized Stored Procedures

Scenario

In this exercise, you will create a stored procedure in the MarketDev database to support one of the new reports.

Supporting Documentation

Stored Procedure	Marketing.GetProductsByColor
Input parameters	@Color (same data type as the Color column in the Production.Product table).
Output parameters	None
Output columns	ProductID, ProductName, ListPrice (returned as a column named Price), Color, Size and SizeUnitMeasureCode (returned as a column named UnitOfMeasure) (from Marketing.Product).
Output order	ProductName
Notes	The procedure should return products that have no Color if the parameter is NULL.

The main tasks for this exercise are as follows:

1. Review the Reports.GetProductsByColor Stored Procedure Specification
2. Design, Create and Test the Reports.GetProductsByColor Stored Procedure

▶ Task 1: Review the Reports.GetProductsByColor Stored Procedure Specification

- Review the design requirements in the Exercise Scenario for Reports.GetProductsByColor.

▶ Task 2: Design, Create and Test the Reports.GetProductsByColor Stored Procedure

1. Design and implement the stored procedure.
2. Execute the stored procedure.

 **Note:** Ensure that approximately 26 rows are returned for blue products. Ensure that approximately 248 rows are returned for products that have no color.

Results: After completing this exercise, you will have:

Created the GetProductsByColor parameterized stored procedure.

Exercise 3: Change Stored Procedure Execution Context

Scenario

In this exercise, you will alter the stored procedures to use a different execution context.

The main tasks for this exercise are as follows:

1. Alter the Reports.GetProductColors Stored Procedure to Execute As OWNER
2. Alter the Reports.GetProductsAndModels Stored Procedure to Execute As OWNER.
3. Alter the Reports.GetProductsByColor Stored Procedure to Execute As OWNER

▶ Task 1: Alter the Reports.GetProductColors Stored Procedure to Execute As OWNER

- Alter the stored procedure Reports.GetProductColors so that it executes as owner.

▶ Task 2: Alter the Reports.GetProductsAndModels Stored Procedure to Execute As OWNER.

- Alter the stored procedure Reports.GetProductsAndModels so that it executes as owner.

▶ Task 3: Alter the Reports.GetProductsByColor Stored Procedure to Execute As OWNER

- Alter the stored procedure, Reports.GetProductsByColor so that it executes as owner.

Results: After completing this exercise, you will have altered the three stored procedures created in earlier exercises, so that they run as owner.

Lab Answer Key 7: Designing and Implementing Stored Procedures

Exercise 1: Create Stored Procedures

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab09\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and wait for the script to finish.

▶ Task 2: Review the Reports.GetProductColors Stored Procedure Specification

- Review the design requirements in the Exercise Scenario for Marketing.GetProductColors.

▶ Task 3: Design, Create and Test the Reports.GetProductColors Stored Procedure

1. Start SQL Server Management Studio, and connect to the **MIA-SQL** database instance using Windows authentication.
2. In Object Explorer, expand **MIA-SQL**, and then expand **Databases**.
3. Right-click the **MarketDev** database, and then click **New Query**.
4. In the query pane, type the query below.

```
CREATE PROCEDURE Reports.GetProductColors
AS
SET NOCOUNT ON;
BEGIN
SELECT DISTINCT p.Color
FROM Marketing.Product AS p
WHERE p.Color IS NOT NULL
ORDER BY p.Color;
END
GO
EXEC Reports.GetProductColors;
GO
```

5. On the toolbar, click **Execute**.
6. **Note**: Ensure that approximately nine colors are returned and that no NULL row is returned.

▶ Task 4: Review the Reports.GetProductsAndModels Stored Procedure Specification

7. Review the supplied design requirements in the supporting documentation in the Exercise Scenario for Reports.GetProductsAndModels.

▶ Task 5: Design, Create and Test the Reports.GetProductsAndModels Stored Procedure

1. In Object Explorer, right-click the **MarketDev** database and then click **New Query**.
2. In the query pane, type the following query:

```
CREATE PROCEDURE Reports.GetProductsAndModels
AS
SET NOCOUNT ON;
BEGIN
SELECT p.ProductID,
       p.ProductName,
       p.ProductNumber,
       p.SellStartDate,
       p.SellEndDate,
       p.Color,
      pm.ProductModelID,
       COALESCE(ed.Description,id.Description,p.ProductName) AS EnglishDescription,
       COALESCE(fd.Description,id.Description,p.ProductName) AS FrenchDescription,
       COALESCE(cd.Description,id.Description,p.ProductName) AS ChineseDescription
FROM Marketing.Product AS p
LEFT OUTER JOIN Marketing.ProductModel AS pm
ON p.ProductModelID = pm.ProductModelID
LEFT OUTER JOIN Marketing.ProductDescription AS ed
ON pm.ProductModelID = ed.ProductModelID
AND ed.LanguageID = 'en'
LEFT OUTER JOIN Marketing.ProductDescription AS fd
ON pm.ProductModelID = fd.ProductModelID
AND fd.LanguageID = 'fr'
LEFT OUTER JOIN Marketing.ProductDescription AS cd
ON pm.ProductModelID = cd.ProductModelID
AND cd.LanguageID = 'zh-cht'
LEFT OUTER JOIN Marketing.ProductDescription AS id
ON pm.ProductModelID = id.ProductModelID
AND id.LanguageID = ''
ORDER BY p.ProductID,pm.ProductModelID;
END
GO
EXEC Reports.GetProductsAndModels;
GO
```

3. On the toolbar, click **Execute**.

Results: After completing this lab, you will have created and tested two stored procedures, Reports.GetProductColors and Reports.GetProductsAndModels.

Exercise 2: Create Parameterized Stored Procedures

▶ Task 1: Review the Reports.GetProductsByColor Stored Procedure Specification

- Review the design requirements in the Exercise Scenario for Reports.GetProductsByColor.

▶ Task 2: Design, Create and Test the Reports.GetProductsByColor Stored Procedure

1. In Object Explorer, right-click the **MarketDev** database and then click **New Query**.
2. In the query pane, type the following query:

```
CREATE PROCEDURE Marketing.GetProductsByColor
@Color nvarchar(16)
AS
SET NOCOUNT ON;
BEGIN
SELECT p.ProductID,
p.ProductName,
p.ListPrice AS Price,
p.Color,
p.Size,
p.SizeUnitMeasureCode AS UnitOfMeasure
FROM Marketing.Product AS p
WHERE (p.Color = @Color) OR (p.Color IS NULL AND @Color IS NULL)
ORDER BY ProductName;
END
GO
```

3. On the toolbar, click **Execute**.
4. In the query pane, under the existing code, type the following Transact-SQL, highlight the query, and then click **Execute**:

```
EXEC Marketing.GetProductsByColor 'Blue';
GO
EXEC Marketing.GetProductsByColor NULL;
GO
```

 **Note:** Ensure that approximately 26 rows are returned for blue products. Ensure that approximately 248 rows are returned for products that have no color.

Results: After completing this exercise, you will have:

Created the GetProductsByColor parameterized stored procedure.

Exercise 3: Change Stored Procedure Execution Context

▶ Task 1: Alter the Reports.GetProductColors Stored Procedure to Execute As OWNER

1. In Object Explorer, right-click the **MarketDev** database and then click **New Query**.
2. In the query pane, type the following query:

```
ALTER PROCEDURE Reports.GetProductColors
WITH EXECUTE AS OWNER
AS
SET NOCOUNT ON;
BEGIN
SELECT DISTINCT p.Color
FROM Marketing.Product AS p
WHERE p.Color IS NOT NULL
ORDER BY p.Color;
END
GO
```

3. On the toolbar, click **Execute**.

▶ Task 2: Alter the Reports.GetProductsAndModels Stored Procedure to Execute As OWNER.

1. In Object Explorer, right-click the **MarketDev** database and then click **New Query**.
2. In the query pane, type the following query:

```
ALTER PROCEDURE Reports.GetProductsAndModels
WITH EXECUTE AS OWNER
AS
SET NOCOUNT ON;
BEGIN
SELECT p.ProductID,
       p.ProductName,
       p.ProductNumber,
       p.SellStartDate,
       p.SellEndDate,
       p.Color,
       pm.ProductModelID,
       COALESCE(ed.Description,id.Description,p.ProductName) AS EnglishDescription,
       COALESCE(fd.Description,id.Description,p.ProductName) AS FrenchDescription,
       COALESCE(cd.Description,id.Description,p.ProductName) AS ChineseDescription
FROM Marketing.Product AS p
LEFT OUTER JOIN Marketing.ProductModel AS pm
ON p.ProductModelID = pm.ProductModelID
LEFT OUTER JOIN Marketing.ProductDescription AS ed
ON pm.ProductModelID = ed.ProductModelID
AND ed.LanguageID = 'en'
LEFT OUTER JOIN Marketing.ProductDescription AS fd
ON pm.ProductModelID = fd.ProductModelID
AND fd.LanguageID = 'fr'
LEFT OUTER JOIN Marketing.ProductDescription AS cd
ON pm.ProductModelID = cd.ProductModelID
AND cd.LanguageID = 'zh-cht'
LEFT OUTER JOIN Marketing.ProductDescription AS id
ON pm.ProductModelID = id.ProductModelID
AND id.LanguageID = ''
ORDER BY p.ProductID,pm.ProductModelID;
END
GO
```

3. On the toolbar, click **Execute**.

▶ Task 3: Alter the Reports.GetProductsByColor Stored Procedure to Execute As OWNER

1. In Object Explorer, right-click the **MarketDev** database, and then click **New Query**.
2. In the query pane, type the following query:

```
ALTER PROCEDURE Marketing.GetProductsByColor
@Color nvarchar(16)
WITH EXECUTE AS OWNER
AS
SET NOCOUNT ON;
BEGIN
SELECT p.ProductID,
p.ProductName,
p.ListPrice AS Price,
p.Color,
p.Size,
p.SizeUnitMeasureCode AS UnitOfMeasure
FROM Marketing.Product AS p
WHERE (p.Color = @Color) OR (p.Color IS NULL AND @Color IS NULL)
ORDER BY ProductName;
END
GO
```

3. On the toolbar, click **Execute**.

Results: After completing this exercise, you will have altered the three stored procedures created in earlier exercises, so that they run as owner.

Lab 8: Designing and Implementing User-Defined Functions

Scenario

The existing marketing application includes some functions. Your manager has requested your assistance in creating a new function for formatting phone numbers—you also need to modify an existing function to improve its usability. You will work in the Adventureworks database.

Objectives

After completing this lab, you will be able to:

- Create a function.
- Modify an existing function.

Virtual machines: **20762C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Format Phone Numbers

Scenario

Your manager has noticed that different users tend to format phone numbers that are entered into the database in different ways. She has asked you to create a function that will be used to format the phone numbers. You need to design, implement, and test the function.

The main tasks for this exercise are as follows:

- Review the design requirements.
- Design and create the function.
- Test the function.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Review the Design Requirements
3. Design and Create the Function
4. Test the Function

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Labfiles\Lab10\Starter\Setup.cmd** as an administrator.

▶ Task 2: Review the Design Requirements

1. Open the **Supporting Documentation.docx** in the **D:\Labfiles\Lab10\Starter** folder.
2. Review the **Function Specifications: Phone Number** section in the supporting documentation.

▶ Task 3: Design and Create the Function

- Design and create the function for reformatting phone numbers.

▶ Task 4: Test the Function

- Execute the **FormatPhoneNumber** function to ensure that the function correctly formats the phone number.

Results: After this exercise, you should have created a new **FormatPhoneNumber** function within the **dbo** schema.

Exercise 2: Modify an Existing Function

Scenario

An existing function, **dbo.StringListToTable**, takes a comma-delimited list of strings and returns a table. In some application code, this causes issues with data types because the list often contains integers rather than just strings.

The main tasks for this exercise are as follows:

1. Review the requirements.
2. Design and create the function.
3. Test the function.
4. Test the function by using an alternate delimiter, such as the pipe character (|).

The main tasks for this exercise are as follows:

1. Review the Requirements
2. Design and Create the Function
3. Test the Function
4. Test the Function by Using an Alternate Delimiter

▶ Task 1: Review the Requirements

- In the **Supporting Documentation.docx**, review the **Requirements: Comma Delimited List Function** section in the supporting documentation.

▶ Task 2: Design and Create the Function

- Design and create the **dbo.IntegerListToTable** function.

▶ Task 3: Test the Function

- Execute the **dbo.IntegerListToTable** function to ensure that it returns the correct results.

▶ Task 4: Test the Function by Using an Alternate Delimiter

- Test the **dbo.IntegerListToTable** function, and then pass in an alternate delimiter, such as the pipe character (|).

Results: After this exercise, you should have created a new **IntegerListToTable** function within a dbo schema.

Lab Answer Key 8: Designing and Implementing User-Defined Functions

Exercise 1: Format Phone Numbers

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab10\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Access Control** dialog box, click **Yes**.
4. Wait for **setup.cmd** to complete successfully.

▶ Task 2: Review the Design Requirements

1. In File Explorer, navigate to **D:\Labfiles\Lab10\Starter**, and then double-click **Supporting Documentation.docx**.
2. Review the **Function Specifications: Phone Number** section in the supporting documentation.

▶ Task 3: Design and Create the Function

1. On the taskbar, click **Microsoft SQL Server Management Studio**.
2. In the **Connect to Server** dialog box, ensure the server name is **MIA-SQL**, and then click **Connect**.
3. In Object Explorer, expand **MIA-SQL**, expand **Databases**, right-click **AdventureWorks**, and then click **New Query**.
4. Type the following query in the query pane:

```
CREATE FUNCTION dbo.FormatPhoneNumber
( @PhoneNumberToFormat nvarchar(16)
)
RETURNS nvarchar(16)
AS BEGIN
DECLARE @Digits nvarchar(16) = '';
DECLARE @Remaining nvarchar(16) = @PhoneNumberToFormat;
DECLARE @Character nchar(1);
IF LEFT(@Remaining,1) = N'+' RETURN @Remaining;
WHILE (LEN(@Remaining) > 0) BEGIN
SET @Character = LEFT(@Remaining,1);
SET @Remaining = SUBSTRING(@Remaining,2,LEN(@Remaining));
IF (@Character >= N'0') AND (@Character <= N'9')
SET @Digits += @Character;
END;
RETURN CASE LEN(@Digits)
WHEN 10 THEN N'(' + SUBSTRING(@Digits,1,3) + N') '
+ SUBSTRING(@Digits,4,3) + N'-'
+ SUBSTRING(@Digits,7,4)
WHEN 8 THEN SUBSTRING(@Digits,1,4) + N'-'
+ SUBSTRING(@Digits,5,4)
WHEN 6 THEN SUBSTRING(@Digits,1,3) + N'-'
+ SUBSTRING(@Digits,4,3)
```

```
ELSE @Digits
END;
END;
GO
```

5. In the toolbar, click **Execute**.

▶ Task 4: Test the Function

1. In Object Explorer, right-click **AdventureWorks**, and then click **New Query**.
2. In the query pane, type the following query:

```
SELECT dbo.FormatPhoneNumber('+61 3 9485-2342');
SELECT dbo.FormatPhoneNumber('415 485-2342');
SELECT dbo.FormatPhoneNumber('(41) 5485-2342');
SELECT dbo.FormatPhoneNumber('94852342');
SELECT dbo.FormatPhoneNumber('85-2342');
GO
```

3. In the toolbar, click **Execute**.

Note:
The output should resemble the following:
+61 3 9485-2342
(415) 485-2342
(415) 485-2342
9485-2342

4. 852-342

Results: After this exercise, you should have created a new **FormatPhoneNumber** function within the **dbo** schema.

Exercise 2: Modify an Existing Function

▶ Task 1: Review the Requirements

- In WordPad, in the **Supporting Documentation.docx**, review the requirement for the **dbo.IntegerListToTable** function in the supporting documentation.

▶ Task 2: Design and Create the Function

1. In SQL Server Management Studio, in Object Explorer, right-click **AdventureWorks**, and then click **New Query**.
2. In the query pane, type the following query:

```
CREATE FUNCTION dbo.IntegerListToTable
( @InputList nvarchar(MAX),@Delimiter nchar(1) = N',')
RETURNS @OutputTable TABLE (PositionInList int IDENTITY(1, 1) NOT NULL,IntegerValue
int)
AS BEGIN
          INSERT INTO @OutputTable (IntegerValue)
          SELECT Value FROM STRING_SPLIT ( @InputList , @Delimiter );
   RETURN;
END;
GO
```

3. In the toolbar, click **Execute**.

▶ Task 3: Test the Function

1. In Object Explorer, right-click the **AdventureWorks** database, and then click **New Query**.
2. In the query pane, type the following query:

```
SELECT * FROM dbo.IntegerListToTable('234,354253,3242,2',',');
GO
```

3. In the toolbar, click **Execute**.

   Note that the output should resemble the following:

PositionInList	IntegerValue
1	234
2	354253
3	3242
4	2

▶ Task 4: Test the Function by Using an Alternate Delimiter

1. In Object Explorer, right-click the **AdventureWorks** database, and then click **New Query**.
2. In the query pane, type the following query:

```
SELECT * FROM dbo.IntegerListToTable('234|354253|3242|2','|');
GO
```

3. In the toolbar, click **Execute**.

   Note that the output should resemble the following:

PositionInList	IntegerValue
1	234
2	354253
3	3242
4	2

4. Close SSMS without saving.

Results: After this exercise, you should have created a new **IntegerListToTable** function within a dbo schema.

Lab 9: Responding to Data Manipulation by Using Triggers

Scenario

You are required to audit any changes to data in a table that contains sensitive balance data. You have decided to implement this by using DML triggers because the SQL Server Audit mechanism does not provide directly for the requirements in this case.

Supporting Documentation

The **Production.ProductAudit** table is used to hold changes to high value products. The data to be inserted in each column is shown in the following table:

Column	Data type	Value to insert
AuditID	int	IDENTITY
ProductID	int	ProductID
UpdateTime	datetime2	SYSDATETIME()
ModifyingUser	varchar(30)	ORIGINAL_LOGIN()
OriginalListPrice	decimal(18,2)	ListPrice before update
NewListPrice	decimal(18,2)	ListPrice after update

Objectives

After completing this lab, you will be able to:

- Create triggers
- Modify triggers
- Test triggers

Virtual Machine: **20762C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Create and Test the Audit Trigger

Scenario

The **Production.Product** table includes a column called **ListPrice**. Whenever an update is made to the table, if either the existing balance or the new balance is greater than **1,000 US dollars**, an entry must be written to the **Production.ProductAudit** audit table.

 **Note:** Inserts or deletes on the table do not have to be audited. Details of the current user can be taken from the ORIGINAL_LOGIN() function.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Review the Design Requirements
3. Design a Trigger
4. Test the Behavior of the Trigger

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab11\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.

▶ Task 2: Review the Design Requirements

1. In the **D:\Labfiles\Lab11\Starter** folder, open the **Supporting Documentation.docx**.
2. In SQL Server Management Studio, review the existing structure of the **Production.ProductAudit** table and the values required in each column, based on the supporting documentation.
3. Review the existing structure of the **Production.Product** table on SSMS.

▶ Task 3: Design a Trigger

- Design and create a trigger that meets the needs of the supporting documentation.

▶ Task 4: Test the Behavior of the Trigger

- Execute data modification statements that are designed to test whether the trigger is working as expected.

Results: After this exercise, you should have created a new trigger. Tests should have shown that it is working as expected.

Exercise 2: Improve the Audit Trigger

Scenario

Now that the trigger created in the first exercise has been deployed to production, the operations team is complaining that too many entries are being audited. Many accounts have more than **10,000 US dollars** as a balance and minor movements of money are causing audit entries. You must modify the trigger so that only changes in the balance of more than **10,000 US dollars** are audited instead.

The main tasks for this exercise are as follows:

1. Modify the trigger based on the updated requirements.
2. Delete all rows from the **Marketing.CampaignAudit** table.
3. Test the modified trigger.

The main tasks for this exercise are as follows:

1. Modify the Trigger
2. Delete all Rows from the Marketing.CampaignAudit Table
3. Test the Modified Trigger

▶ Task 1: Modify the Trigger

1. Review the design of the existing trigger and decide what modifications are required.
2. Use an ALTER TRIGGER statement to change the existing trigger so that it will meet the updated requirements.

▶ Task 2: Delete all Rows from the Marketing.CampaignAudit Table

- Execute a DELETE statement to remove all existing rows from the **Marketing.CampaignAudit** table.

▶ Task 3: Test the Modified Trigger

1. Execute data modification statements that are designed to test whether the trigger is working as expected.
2. Close SQL Server Management Studio without saving anything.

Results: After this exercise, you should have altered the trigger. Tests should show that it is now working as expected.

Lab Answer Key 9: Responding to Data Manipulation Via Triggers

Exercise 1: Create and Test the Audit Trigger

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and **20762C-MIA-SQL** virtual machines are both running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab11\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Access Control** dialog box, click **Yes** and wait for the script to complete.

▶ Task 2: Review the Design Requirements

1. Using File Explorer, browse to **D:\Labfiles\Lab11\Starter**.
2. Double-click **Supporting Documentation.docx**, review the supplied table requirements in the supporting documentation for the **Production.ProductAudit** table.
3. On the taskbar, click **Microsoft SQL Server Management Studio**.
4. In the **Connect to Server** dialog box, ensure that the **Server name** is **MIA-SQL**, and then click **Connect**.
5. In Object Explorer, under **MIA-SQL**, expand **Databases**, expand **AdventureWorks**, expand **Tables**, expand **Production.Product**, and then expand **Columns**.
6. Review the table design.
7. Do not close SQL Server Management Studio.

▶ Task 3: Design a Trigger

1. Click **New Query**.
2. In the query pane, type the following query:

```
USE AdventureWorks
GO
CREATE TABLE Production.ProductAudit(
ProductID int NOT NULL,
UpdateTime datetime2(7) NOT NULL,
ModifyingUser nvarchar(100) NOT NULL,
OriginalListPrice money NULL,
NewListPrice money NULL
)
GO
CREATE TRIGGER Production.TR_ProductListPrice_Update
ON Production.Product
AFTER UPDATE
AS BEGIN
   SET NOCOUNT ON;
   INSERT Production.ProductAudit(ProductID, UpdateTime, ModifyingUser,
OriginalListPrice,NewListPrice)
   SELECT Inserted.ProductID,SYSDATETIME(),ORIGINAL_LOGIN(),deleted.ListPrice,
inserted.ListPrice
   FROM deleted
```

```
   INNER JOIN inserted
   ON deleted.ProductID = inserted.ProductID
   WHERE deleted.ListPrice > 1000 OR inserted.ListPrice > 1000;
END;
GO
```

3. On the toolbar, click **Execute**.
4. Do not close SQL Server Management Studio.

▶ Task 4: Test the Behavior of the Trigger

1. Click **New Query**.
2. In the query pane, type the following query:

```
UPDATE Production.Product
SET ListPrice=3978.00
WHERE ProductID BETWEEN 749 and 753;
GO
SELECT * FROM Production.ProductAudit;
GO
```

3. On the toolbar, click **Execute**.
4. Note: there should be five rows in the **Production.ProductAudit** table.
5. Do not close SQL Server Management Studio.

Results: After this exercise, you should have created a new trigger. Tests should have shown that it is working as expected.

Exercise 2: Improve the Audit Trigger

▶ Task 1: Modify the Trigger

1. In Object Explorer, under **Databases**, right-click the **MarketDev** database, and then click **Refresh**.
2. Expand the **MarketDev** database, expand **Tables**, expand **Marketing.CampaignBalance**, and then expand **Triggers**.
3. Double-click **TR_CampaignBalance_Update**, and review the trigger design in the query pane.
4. Click **New Query**.
5. In the query pane, type the following query:

```
USE MarketDev
GO
ALTER TRIGGER Marketing.TR_CampaignBalance_Update
ON Marketing.CampaignBalance
AFTER UPDATE
AS BEGIN
SET NOCOUNT ON;
INSERT Marketing.CampaignAudit
(AuditTime, ModifyingUser, RemainingBalance)
SELECT SYSDATETIME(),
ORIGINAL_LOGIN(),
inserted.RemainingBalance
FROM deleted
INNER JOIN inserted
ON deleted.CampaignID = inserted.CampaignID
WHERE ABS(deleted.RemainingBalance - inserted.RemainingBalance) > 10000;
END;
GO
```

6. In the toolbar, click **Execute**.
7. Do not close SQL Server Management Studio.

▶ Task 2: Delete all Rows from the Marketing.CampaignAudit Table

1. Click **New Query**.
2. In the query pane, type the following query:

```
DELETE FROM Marketing.CampaignAudit;
GO
```

3. In the toolbar, click **Execute**.
4. Do not close SQL Server Management Studio.

▶ Task 3: Test the Modified Trigger

1. Click **New Query**.
2. In the query pane, type the following query:

```
SELECT * FROM Marketing.CampaignBalance;
GO
EXEC Marketing.MoveCampaignBalance 3,2,10100;
GO
EXEC Marketing.MoveCampaignBalance 3,2,1010;
GO
SELECT * FROM Marketing.CampaignAudit;
GO
```

3. In the toolbar, click **Execute**.
4. Close SQL Server Management Studio without saving anything.

Results: After this exercise, you should have altered the trigger. Tests should show that it is now working as expected.

Lab 10: Concurrency and Transactions

Scenario

You have reviewed statistics for the **AdventureWorks** database and noticed high wait stats for CPU, memory, IO, blocking, and latching. In this lab, you will address blocking wait stats. You will explore workloads that can benefit from snapshot isolation and partition level locking. You will then implement snapshot isolation and partition level locking to reduce overall blocking.

Objectives

After completing this lab, you will be able to:

- Implement the SNAPSHOT isolation level.
- Implement partition level locking.

Virtual machine: **20762C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Implement Snapshot Isolation

Scenario

You have reviewed wait statistics for the **AdventureWorks** database and noticed high wait stats for locking, amongst others. In this exercise, you will implement SNAPSHOT isolation to reduce blocking scenarios.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Clear Wait Statistics
3. Run the Workload
4. Capture Lock Wait Statistics
5. Enable SNAPSHOT Isolation
6. Implement SNAPSHOT Isolation
7. Rerun the Workload
8. Capture New Lock Wait Statistics
9. Compare Overall Lock Wait Time

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **20762C-MIA-DC**, and **20762C-MIA-SQL** virtual machines are running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab17\Starter** folder as Administrator.

▶ Task 2: Clear Wait Statistics

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** instance using Windows authentication, then open the project file **D:\Labfiles\Lab17\Starter\Project\Project.ssmssln** and the script file **Lab Exercise 01 - snapshot isolation.sql**.
2. Execute the query under the comment that begins **Task 1** to clear wait statistics.

▶ Task 3: Run the Workload

- In the **D:\Labfiles\Lab17\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell™. Wait for the workload to finish before continuing. If a message is displayed asking you to confirm a change in execution policy, type **Y** and then press Enter.

▶ Task 4: Capture Lock Wait Statistics

- In SSMS, amend the query under the comment that begins **Task 3** to capture only lock wait statistics into a temporary table. Hint: lock wait statistics have a **wait_type** that begins "LCK".

▶ Task 5: Enable SNAPSHOT Isolation

- Amend the properties of the **AdventureWorks** database to allow SNAPSHOT isolation.

▶ Task 6: Implement SNAPSHOT Isolation

1. In SSMS Solution Explorer, open the script file **Lab Exercise 01 - stored procedure.sql**.
2. Use the script to modify the stored procedure definition to run under SNAPSHOT isolation.

▶ Task 7: Rerun the Workload

1. In the SSMS query window for **Lab Exercise 01 - snapshot isolation.sql,** rerun the query under the comment that begins **Task 1**.
2. In the **D:\Labfiles\Lab17\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell. Wait for the workload to finish before continuing.

▶ Task 8: Capture New Lock Wait Statistics

- In SSMS, under the comment that begins **Task 8**, amend the query to capture lock wait statistics into a temporary table called **#task8**.

▶ Task 9: Compare Overall Lock Wait Time

- In the SSMS query window for **Lab Exercise 01 - snapshot isolation.sql**, execute the query under the comment that begins **Task 9**, to compare the total **wait_time_ms** you have captured between the **#task3** and **#task8** temporary tables.

Results: After this exercise, the **AdventureWorks** database will be configured to use the SNAPSHOT isolation level.

Exercise 2: Implement Partition Level Locking

Scenario

You have reviewed statistics for the **AdventureWorks** database and noticed high wait stats for locking, amongst others. In this exercise, you will implement partition level locking to reduce blocking.

The main tasks for this exercise are as follows:

1. Open Activity Monitor
2. Clear Wait Statistics
3. View Lock Waits in Activity Monitor
4. Enable Partition Level Locking
5. Rerun the Workload

▶ Task 1: Open Activity Monitor

1. In SSMS Object Explorer, open Activity Monitor for the **MIA-SQL** instance.
2. In Activity Monitor, expand the **Resource Waits** section.

▶ Task 2: Clear Wait Statistics

1. If it is not already open, open the project file **D:\Labfiles\Lab17\Starter\Project\Project.ssmssln**, then open the query file **Lab Exercise 02 - partition isolation.sql**.
2. Execute the code under **Task 2** to clear wait statistics.

▶ Task 3: View Lock Waits in Activity Monitor

1. In the **D:\Labfiles\Lab17\Starter** folder, execute **start_load_exercise_02.ps1** with PowerShell. Wait for the workload to finish before continuing (it will take a few minutes to complete).
2. Switch to SSMS and to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.
3. Close the PowerShell window where the workload was executed.

▶ Task 4: Enable Partition Level Locking

1. In the **Lab Exercise 02 - partition isolation.sql** query, under the comment that begins **Task 5**, write a query to alter the **Proseware.CampaignResponsePartitioned** table in the **AdventureWorks** database to enable partition level locking.
2. Rerun the query under the comment that begins **Task 2** to clear wait statistics.

▶ Task 5: Rerun the Workload

1. In the **D:\Labfiles\Lab17\Starter** folder, execute **start_load_exercise_02.ps1** with PowerShell. Wait for the workload to finish before continuing (it will take a few minutes to complete).
2. Return to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec**) for the **Lock** wait type.
3. Compare this value to the value you noted earlier in the exercise.
4. Close the PowerShell window where the workload was executed.
5. Close SQL Server Management Studio without saving any changes.

Results: After this exercise, the **AdventureWorks** database will use partition level locking.

Lab Answer Key 10: Concurrency and Transactions

Exercise 1: Implement Snapshot Isolation

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **20762C-MIA-DC**, and **20762C-MIA-SQL** virtual machines are running, and then log on to **20762C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab17\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Clear Wait Statistics

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab17\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click the query **Lab Exercise 01 - snapshot isolation.sql**.
5. To clear wait statistics, select the query under the comment that begins **Task 1**, and then click **Execute**.

▶ Task 3: Run the Workload

1. Open File Explorer, navigate to the **D:\Labfiles\Lab17\Starter** folder.
2. Right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**.
3. If a message is displayed asking you to confirm a change in execution policy, type **Y**, and then press Enter.
4. Wait for the workload to complete, and then press Enter to close the window.

▶ Task 4: Capture Lock Wait Statistics

1. In SQL Server Management Studio, in the query pane, edit the query under the comment that begins **Task 3** so that it reads:

```
SELECT wait_type, waiting_tasks_count, wait_time_ms,
max_wait_time_ms, signal_wait_time_ms
INTO #task3
FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'LCK%'
AND wait_time_ms > 0
ORDER BY wait_time_ms DESC;
```

2. Select the query you have amended and click **Execute**.

▶ Task 5: Enable SNAPSHOT Isolation

1. In SQL Server Management Studio Object Explorer, under MIA-SQL, expand **Databases**, right-click **AdventureWorks**, and then click **Properties**.
2. In the **Database Properties - AdventureWorks** dialog box, on the **Options** page, in the **Miscellaneous** section, change the value of the **Allow Snapshot Isolation** setting to **True**, and then click **OK**.

▶ Task 6: Implement SNAPSHOT Isolation

1. In Solution Explorer, double-click the query **Lab Exercise 01 - stored procedure.sql**.
2. Amend the stored procedure definition in the file so that it reads:

```
USE AdventureWorks;
GO
ALTER PROC Proseware.up_Campaign_Report
AS
   SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
   SELECT TOP 10 * FROM Sales.SalesTerritory AS T
   JOIN (
          SELECT CampaignTerritoryID,
          DATEPART(MONTH, CampaignStartDate) as start_month_number,
          DATEPART(MONTH, CampaignEndDate) as end_month_number,
          COUNT(*) AS campaign_count
          FROM Proseware.Campaign
          GROUP BY CampaignTerritoryID, DATEPART(MONTH,
CampaignStartDate),DATEPART(MONTH, CampaignEndDate)
   ) AS x
   ON x.CampaignTerritoryID = T.TerritoryID
   ORDER BY campaign_count;
GO
```

3. Highlight the script and click **Execute**.

▶ Task 7: Rerun the Workload

1. In the SQL Server Management Studio query window for **Lab Exercise 01 - snapshot isolation.sql**, select the query under the comment that begins **Task 1**, and then click **Execute**.
2. In File Explorer, in the **D:\Labfiles\Lab17\Starter** folder, right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**.
3. Wait for the workload to complete.

▶ Task 8: Capture New Lock Wait Statistics

1. In SQL Server Management Studio, in the query window for **Lab Exercise 01 - snapshot isolation.sql**, amend the query under the comment that begins **Task 8** so that it reads:

```
SELECT wait_type, waiting_tasks_count, wait_time_ms,
max_wait_time_ms, signal_wait_time_ms
INTO #task8
FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'LCK%'
AND wait_time_ms > 0
ORDER BY wait_time_ms DESC;
```

2. Select the query you have amended and click **Execute**.

▶ Task 9: Compare Overall Lock Wait Time

1. In SQL Server Management Studio, in the query pane, select the query under the comment that begins **Task 9** and click **Execute**. Compare the total **wait_time_ms** you have captured between the **#task3** and **#task8** temporary tables. Note that the wait time in the **#task8** table—after SNAPSHOT isolation was implemented—is lower.
2. Close the query windows for **Lab Exercise 01 - snapshot isolation.sql** and **Lab Exercise 01 - stored procedure.sql** without saving changes, but leave SQL Server Management Studio open for the next exercise.

Results: After this exercise, the **AdventureWorks** database will be configured to use the SNAPSHOT isolation level.

Exercise 2: Implement Partition Level Locking

▶ Task 1: Open Activity Monitor

1. In SQL Server Management Studio, in Object Explorer, right-click **MIA-SQL** and click **Activity Monitor**.
2. In Activity Monitor, click **Resource Waits** to expand the section.

▶ Task 2: Clear Wait Statistics

1. In Solution Explorer, double-click **Lab Exercise 02 - partition isolation.sql**.
2. Select the code under the comment that begins **Task 2**, and click **Execute**.

▶ Task 3: View Lock Waits in Activity Monitor

1. In File Explorer, in the **D:\Labfiles\Lab17\Starter** folder, right-click **start_load_exercise_02.ps1**, and then click **Run with PowerShell**.
2. Wait for the workload to complete (it will take a few minutes).
3. In SQL Server Management Studio, on the **MIA-SQL - Activity Monitor** tab, in the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.
4. In the PowerShell workload window, press Enter to close it.

▶ Task 4: Enable Partition Level Locking

1. In SQL Server Management Studio, in the **Lab Exercise 02 - partition isolation.sql** query, under the comment that begins **Task 5**, type:

```
USE AdventureWorks;
GO
ALTER TABLE Proseware.CampaignResponsePartitioned SET (LOCK_ESCALATION = AUTO);
GO
```

2. Select the query you have typed and click **Execute**.
3. Select the query under the comment that begins **Task 2**, then click **Execute**.

▶ Task 5: Rerun the Workload

1. In File Explorer, in the **D:\Labfiles\Lab17\Starter** folder, right-click **start_load_exercise_02.ps1**, and then click **Run with PowerShell**.
2. Wait for the workload to complete (it will take a few minutes).
3. In SQL Server Management Studio, on the **MIA-SQL - Activity Monitor** tab, in the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.
4. Compare this value to the value you noted earlier in the exercise; the wait time will be considerably lower after you implemented partition level locking.
5. In the PowerShell workload window, press Enter to close it.
6. Close SQL Server Management Studio without saving any changes.

Results: After this exercise, the **AdventureWorks** database will use partition level locking.