Lab Manual

T2762-2, Developing SQL Databases, Part 2

Lab 1: Advanced table designs

Scenario

You are a database developer for Adventure Works who will be designing solutions using corporate databases stored in SQL Server. You have been provided with a set of business requirements and will implement partitioning to archive older data, and data compression to obtain optimal performance and storage from your tables.

Objectives

After completing the lab exercises, you will be able to:

- Create partitioned tables.
- Compress data in tables.

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Partitioning Data

Scenario

You have created the tables for your business analyst, but believe that the solution could be improved by implementing additional functionality. You will implement partitioned tables to move historical content to an alternative partition.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Create the HumanResources Database

3. Implement a Partitioning Strategy

4. Test the Partitioning Strategy

▶ 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\Lab03\Starter** folder as Administrator.

▶ Task 2: Create the HumanResources Database

1. Using SSMS, connect to **MIA-SQL** using Windows authentication.

2. Open the project file **D:\Labfiles\Lab03\Starter\Project\Project\Project.ssmssln**, and the T-SQL script **Lab Exercise 1.sql**. Ensure that you are connected to the **TSQL** database.

3. Create the **HumanResources** database.

▶ Task 3: Implement a Partitioning Strategy

1. In Solution Explorer, open the query Lab Exercise 2.sql.

2. Create four filegroups for the **HumanResources** database: **FG0**, **FG1**, **FG2**, **FG3**.

3. Create a partition function named **pfHumanResourcesDates** in the **HumanResources** database to partition the data by dates.
4. Using the partition function, create a partition scheme named **psHumanResources** to use the four filegroups.
5. Create a **Timesheet** table that will use the new partition scheme.
6. Insert some data into the **Timesheet** table.

▶ Task 4: Test the Partitioning Strategy

1. In Solution Explorer, open the query **Lab Exercise 3.sql**.
2. Type and execute a Transact-SQL SELECT statement that returns all of the rows from the **Timesheet** table, along with the partition number for each row. You can use the **$PARTITION** function to achieve this.
3. Type and execute a Transact-SQL statement to view partition metadata.
4. Create a staging table called **Timesheet_Staging** on **FG1**. This should be identical to the **Timesheet** table.
5. Add a check constraint to the **Timesheet_Staging** table to ensure that the values in the **RegisteredStartTime** column meet the following criteria:
   - All values must be greater than or equal to 2011-10-01 00:00.
   - All values must be less than 2012-01-01 00:00.
   - No values can be NULL.
6. Type a Transact-SQL statement to switch out the data in the partition on the filegroup **FG1** to the table **Timesheet_Staging**. Use the **$PARTITION** function to retrieve the partition number.
7. View the metadata for the partitioned table again to see the changes, and then write and execute a SELECT statement to view the rows in the **Timesheet_Staging** table.
8. Type a Transact-SQL statement to merge the first two partitions, using the value **2011-10-01 00:00**.
9. View the metadata for the partitioned table again to see the changes.
10. Type a Transact-SQL statement to make **FG1** the next used filegroup for the partition scheme.
11. Type a Transact-SQL statement to split the first empty partition, using the value **2012-07-01 00:00**.
12. Type and execute a Transact-SQL statement to add two rows for the new period.
13. View the metadata for the partitioned table again to see the changes.

Results: At the end of this lab, the timesheet data will be partitioned to archive old data.

Exercise 2: Compressing Data

Scenario

The business analyst is satisfied with the partitioning concept you have applied to the Timesheet table. To put the table into production, you will rework the partition to widen the time boundaries to accommodate data over a number of years. After you have populated the Timesheet table, you will decide which compression type to apply to each partition.

The main tasks for this exercise are as follows:

1. Create Timesheet Table for Compression
2. Analyze Storage Savings with Compression
3. Compress Partitions

▶ Task 1: Create Timesheet Table for Compression

1. In Solution Explorer, open the query **Lab Exercise 4.sql**.
2. Type and execute a T-SQL SELECT statement that drops the **Payment.Timesheet** table.
3. Type and execute a T-SQL SELECT statement that drops the **psHumanResources** partition scheme.
4. Type and execute a T-SQL SELECT statement that drops the **pfHumanResourcesDates** partition function.
5. Type and execute a T-SQL SELECT statement that creates the **pfHumanResourcesDates** partition function, using **RANGE RIGHT** for the values: **2012-12-31 00:00:00.000**, **2014-12-31 00:00:00.000**, and **2016-12-31 00:00:00.000**.
6. Type and execute a T-SQL SELECT statement that creates the **pfHumanResourcesDates** partition scheme, using the filegroups **FG0**, **FG2**, **FG3**, and **FG1**.
7. Type and execute a T-SQL SELECT statement that creates a **Payment.Timesheet** table, using the **pfHumanResourcesDates** partition scheme.
8. Type and execute a T-SQL SELECT statement that adds staff to three shifts, over the course of six years. Exclude weekend dates.

▶ Task 2: Analyze Storage Savings with Compression

1. In Solution Explorer, open the query **Lab Exercise 5.sql**.
2. Type and execute a T-SQL SELECT statement to view the partition metadata for the **Payment.Timesheet** table.
3. Type and execute a T-SQL SELECT statement to view the estimated savings when applying **ROW** and **PAGE** compression on the **Payment.Timesheet** table.

▶ Task 3: Compress Partitions

1. In Solution Explorer, open the query **Lab Exercise 6.sql**.
2. Type and execute a T-SQL SELECT statement to partition the **Payment.Timesheet** table. Partitions **FG0** and **FG1** should use **ROW** compression, and partitions **FG2** and **FG3** should use **PAGE** compression.
3. Close SSMS without saving anything.

Results: At the end of this lab, the Timesheet table will be populated with six years of data, and will be partitioned and compressed.

Lab 1 Answer Key: Advanced table designs

Exercise 1: Partitioning Data

▶ 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\Lab03\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then if prompted with the question **Do you want to continue with this operation?**, type **Y**, then press Enter.

▶ Task 2: Create the HumanResources Database

1. On the taskbar, click **Microsoft SQL Server Management Studio**.
2. In the **Connect to Server** dialog box, in the **Server name** text box, type **MIA-SQL**, and then click **Options**.
3. In the **Connect to database** list, click **<Browse server...>**.
4. In the **Browse for Databases** dialog box, click **Yes**.
5. In the **Browse Server for Database** dialog box, under **User Databases**, click the **TSQL** database, and then click **OK**.
6. In the **Connect to Server** dialog box, on the **Login** tab, in the **Authentication** list, click **Windows Authentication**, and click **Connect**.
7. On the **File** menu, point to **Open**, and click **Project/Solution**.
8. In the **Open Project** dialog box, navigate to **D:\Labfiles\Lab03\Starter\Project\Project**, and then double-click **Project.ssmssln**.
9. In Solution Explorer, double-click the **Lab Exercise 1.sql** query.
10. When the query window opens, review the code, and click **Execute**.

▶ Task 3: Implement a Partitioning Strategy

1. In Solution Explorer, double-click the query **Lab Exercise 2.sql**.
2. In the query window, under the comment **Create filegroups**, review the Transact-SQL statements, highlight the statements through to the very last line, and then click **Execute**.
3. In the query window, under the comment **Create partition function**, review the Transact-SQL statements, highlight the statements, and then click **Execute**.
4. In the query window, under the comment **Create partition scheme**, review the Transact-SQL statements, highlight the statements, and then click **Execute**.
5. In the query window, under the comment **Create the Timesheet table**, review the Transact-SQL statements, highlight the statements, and then click **Execute**.
6. In the query window, under the comment **Insert data into Timesheet table**, review the Transact-SQL statements, highlight the statements, and then click **Execute**.

▶ Task 4: Test the Partitioning Strategy

1. In Solution Explorer, open the query **Lab Exercise 3.sql**.
2. In the query window, under the comment **Query the Timesheet table**, review the T-SQL statements, highlight the statements, and then click **Execute**.
3. In the query window, under the comment **View partition metadata**, review the T-SQL statements, highlight the statements, and then click **Execute**.
4. In the query window, under the comment **Create the staging table**, review the T-SQL statements, highlight the statements, and then click **Execute**.
5. In the query window, under the comment **Add check constraint**, review the T-SQL statements, highlight the statements, and then click **Execute**.
6. In the query window, under the comment **Switch out the old data**, review the T-SQL statements, highlight the statements, and then click **Execute**.
7. In the query window, under the comment **View archive data in staging table**, review the T-SQL statements, highlight the statements, and then click **Execute**.
8. In the query window, under the comment **View partition metadata**, review the T-SQL statements, highlight the statements, and then click **Execute**.
9. In the query window, under the comment **Merge the first two partitions**, review the T-SQL statements, highlight the statements, and then click **Execute**.
10. In the query window, under the comment **View partition metadata**, review the T-SQL statements, highlight the statements, and then click **Execute**.
11. In the query window, under the comment **Make FG1 the next used filegroup**, review the T-SQL statements, highlight the statements, and then click **Execute**.
12. In the query window, under the comment **Split the empty partition at the end**, review the T-SQL statements, highlight the statements, and then click **Execute**.
13. In the query window, under the comment **Insert data for the new period**, review the T-SQL statements, highlight the statements, and then click **Execute**.
14. In the query window, under the comment **View partition metadata**, review the T-SQL statements, highlight the statements, and then click **Execute**.
15. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this lab, the timesheet data will be partitioned to archive old data.

Exercise 2: Compressing Data

▶ Task 1: Create Timesheet Table for Compression

1. In Solution Explorer, double-click the query **Lab Exercise 4.sql**.
2. In the query window, under the comment **Drop the Payment.Timesheet table**, review the Transact-SQL statements, highlight the statement, and then click **Execute**.
3. In the query window, under the comment **Drop the current partition scheme**, review the Transact-SQL statements, highlight the statement, and then click **Execute**.
4. In the query window, under the comment **Drop the current partition function**, review the Transact-SQL statements, highlight the statement, and then click **Execute**.
5. In the query window, under the comment **Create the new partition function with wider date ranges**, review the Transact-SQL statements, highlight the statement, and then click **Execute**.
6. In the query window, under the comment **Re-create the partition scheme**, review the Transact-SQL statements, highlight the statement, and then click **Execute**.
7. In the query window, under the comment **Create the Timesheet table**, review the Transact-SQL statements, highlight the statement, and then click **Execute**.
8. In the query window, under the comment **Insert data into the Payment.Timesheet table**, review the Transact-SQL statements, highlight the statements through to the final line, and then click **Execute**.

▶ Task 2: Analyze Storage Savings with Compression

1. In Solution Explorer, open the query **Lab Exercise 5.sql**.
2. In the query window, under the comment **View partition metadata**, review the Transact-SQL statements, highlight the statement, and then click **Execute**.
3. In the query window, under the comment **View compression estimated savings**, review the Transact-SQL statements, highlight the statement, and then click **Execute**.

▶ Task 3: Compress Partitions

1. In Solution Explorer, open the query **Lab Exercise 6.sql**.
2. In the query window, review the Transact-SQL statements, and then click **Execute**.
3. Close SSMS without saving anything.

Results: At the end of this lab, the Timesheet table will be populated with six years of data, and will be partitioned and compressed.

Lab 2: Columnstore indexes

Scenario

Adventure Works has created a data warehouse for analytics processing of its current online sales business. Due to large business growth, existing analytical queries are no longer performing as required. Disk space is also becoming more of an issue.

You have been tasked with optimizing the existing database workloads and, if possible, reducing the amount of disk space being used by the data warehouse.

Objectives

After completing this lab, you will be able to:

- Create clustered and nonclustered columnstore indexes.
- Examine an execution plan to check the performance of queries.
- Convert disk based tables into memory optimized tables.

Lab Setup

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Dropping and recreating indexes can take time, depending on the performance of the lab machines.

Exercise 1: Create a Columnstore Index on the FactProductInventory Table

Scenario

You plan to improve the performance of the **AdventureWorksDW** data warehouse by using columnstore indexes. You need to improve the performance of queries that use the **FactProductInventory** tables without causing any database downtime, or dropping any existing indexes. Disk usage for this table is not an issue.

You must retain the existing indexes on the **FactProductInventory** table, and ensure you do not impact current applications by any alterations you make.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Examine the Existing Size of the FactProductInventory Table and Query Performance

3. Create a Columnstore Index on the FactProductInventory 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 MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

2. In the **D:\Labfiles\Lab07\Starter** folder, run **Setup.cmd** as Administrator.

▶ Task 2: Examine the Existing Size of the FactProductInventory Table and Query Performance

1. In SQL Server Management Studio, in the **D:\Labfiles\Lab07\Starter** folder, open the **Query FactProductInventory.sql** script file.
2. Configure SQL Server Management Studio to include the actual execution plan.
3. Execute the script against the **AdventureWorksDW** database. Review the execution plan, making a note of the indexes used, the execution time, and disk space used.

▶ Task 3: Create a Columnstore Index on the FactProductInventory Table

1. Based on the scenario for this exercise, decide whether a clustered or nonclustered columnstore index is appropriate for the **FactProductInventory** table.
2. Create the required columnstore index. Re-execute the query to verify that the new columnstore index is used, along with existing indexes.
3. What, if any, are the disk space and query performance improvements?

Results: After completing this exercise, you will have created a columnstore index and improved the performance of an analytical query. This will have been done in real time without impacting transactional processing.

Exercise 2: Create a Columnstore Index on the FactInternetSales Table

Scenario

You need to improve the performance of queries that use the **FactInternetSales** table. The table has also become large and there are concerns over the disk space being used. You can use scheduled downtime to amend the table and its existing indexes.

Due to existing processing requirements, you must retain the foreign keys on the **FactInternetSales** table, but you can add any number of new indexes to the table.

The main tasks for this exercise are as follows:

1. Examine the Existing Size of the FactInternetSales Table and Query Performance
2. Create a Columnstore Index on the FactInternetSales Table

▶ Task 1: Examine the Existing Size of the FactInternetSales Table and Query Performance

1. In SQL Server Management Studio, in the **D:\Labfiles\Lab07\Starter** folder, open the **Query FactInternetSales.sql** script file.
2. Configure SQL Server Management Studio to include the actual execution plan.
3. Execute the script against the **AdventureWorksDW** database. Review the execution plan, making a note of the indexes used, the execution time, and disk space used.

▶ Task 2: Create a Columnstore Index on the FactInternetSales Table

1. Based on the scenario for this exercise, decide whether a clustered or nonclustered columnstore index is appropriate for the **FactInternetSales** table.
2. Create the required columnstore index. Depending on your chosen index, you may need to drop and recreate keys on the table.

3. Re-execute the query to verify that the new columnstore index is used, along with the existing indexes.
4. What, if any, are the disk space and query performance improvements?

Results: After completing this exercise, you will have greatly reduced the disk space taken up by the FactInternetSales table, and improved the performance of analytical queries against the table.

Optional Exercise 3: Create a Memory Optimized Columnstore Table

Scenario

Due to the improved performance and reduced disk space that columnstore indexes provide, you have been tasked with taking the FactInternetSales table from disk and into memory.

The main tasks for this exercise are as follows:

1. Use the Memory Optimization Advisor
2. Enable the Memory Optimization Advisor to Create a Memory Optimized FactInternetSales Table
3. Examine the Performance of the Memory Optimized Table

▶ Task 1: Use the Memory Optimization Advisor

1. In SQL Server Management Studio, run the **Memory Optimization Advisor** on the **FactInternetSales** table.
2. Note that there are several issues that need to be resolved before the Memory Optimization Advisor can automatically convert the table.

▶ Task 2: Enable the Memory Optimization Advisor to Create a Memory Optimized FactInternetSales Table

1. Using either SQL Server Management Studio, or Transact-SQL statements, drop all the foreign keys and the clustered columnstore index.
2. Memory optimized tables cannot have more than eight indexes. Choose another three indexes to drop.

Note: Hint: consider the rows being used in the **Query FactInternetSales.sql** to guide your decision.

3. Use **Memory Optimization Advisor** on the **FactInternetSales** table.
4. Instead of running the migration with the wizard, script the results for the addition of a columnstore index.

Note: The Memory Optimization Advisor won't suggest columnstore indexes as they are not applicable in all situations. Therefore, these have to be added manually.

5. Note the statements to create a memory optimized filegroup, and the code to copy the existing data.
6. Add a clustered columnstore index to the create table script.
7. Run the edited Transact-SQL.

▶ Task 3: Examine the Performance of the Memory Optimized Table

1. In SQL Server Management Studio, in the **D:\Labfiles\Lab07\Starter** folder, open the **Query FactProductInventory.sql** script file.
2. Configure SQL Server Management Studio to include the actual execution plan.
3. Execute the script against the **AdventureWorksDW** database, and then review the disk space used the execution plan.

Results: After completing this exercise, you will have created a memory optimized version of the **FactInternetSales** disk based table, using the Memory Optimization Advisor.

Lab 2 Answer Key: Columnstore indexes

Exercise 1: Create a Columnstore Index on the FactProductInventory 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 MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Using File Explorer, navigate to the **D:\Labfiles\Lab07\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. When you are prompted, click **Yes** to confirm that you want to run the command file, and then wait for the script to finish.
4. Press any key to continue.

▶ Task 2: Examine the Existing Size of the FactProductInventory Table and Query Performance

1. On the taskbar, click **Microsoft SQL Server Management Studio**.
2. In the **Connect to Server** window, ensure the **Server name** box contains **MIA-SQL** and the **Authentication** box contains **Windows Authentication**, and then click **Connect**.
3. On the **File** menu, point to **Open**, then click **File**.
4. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab07\Starter** folder, click **Query FactProductInventory.sql**, and then click **Open**.
5. Highlight the Transact-SQL under the Step 1 comment, and then click **Execute**.
6. Make a note of the disk space used.
7. On the **Query** menu, click **Include Actual Execution Plan**.
8. Highlight the code after the Step 2 comment, click **Execute**, and then view the query results.
9. On the **Messages** tab, note the CPU and elapsed times.
10. On the **Execution plan** tab, view the execution plan for the query. Position the cursor on the icons from right to left to examine their details, and then note the indexes used. Also, note that the query processor has identified that adding a missing index could improve performance.

▶ Task 3: Create a Columnstore Index on the FactProductInventory Table

1. On the toolbar, click **New Query**, and then in the SQLQuery1.sql pane, type the following Transact-SQL code to create a columnstore index on the **FactProductInventory** table:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX NCI_FactProductInventory_UnitCost_UnitsOut ON
FactProductInventory
(
    ProductKey,
    DateKey,
    UnitCost,
    UnitsOut
)
GO
```

2. On the toolbar, click **Execute** to create the index.

3. Switch back to the **Query FactProductInventory.sql** tab, and then click **Execute** to rerun the query.
4. On the **Execution plan** tab, view the execution plan that is used for the query. Examine the icons from right to left, noting the indexes that were used. Note that the new columnstore index is used.
5. Note the **Actual Execution Mode** is **Batch** instead of **Row** and that this mode is on most of the steps. This is the main reason for the query performance improvement.
6. What, if any, are the disk space and query performance improvements?
7. Close both query windows without saving any changes.

Results: After completing this exercise, you will have created a columnstore index and improved the performance of an analytical query. This will have been done in real time without impacting transactional processing.

Exercise 2: Create a Columnstore Index on the FactInternetSales Table

▶ Task 1: Examine the Existing Size of the FactInternetSales Table and Query Performance

1. On the **File** menu, point to **Open**, then click **File**.
2. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab07\Starter** folder, click **Query FactInternetSales.sql**, and then click **Open**.
3. Highlight the Transact-SQL under the Step 1 comment, and then click **Execute**.
4. Make a note of the disk space used.
5. On the **Query** menu, click **Include Actual Execution Plan**.
6. Highlight the code after the Step 2 comment.
7. On the toolbar, click **Execute**, and then view the query results.
8. On the **Messages** tab, note the CPU and elapsed times.
9. On the **Execution** plan tab, view the execution plan for the query. Position the cursor on the icons from right to left to examine their details and note the indexes that were used. Also note that the query processor has identified that adding a missing index could improve performance.

▶ Task 2: Create a Columnstore Index on the FactInternetSales Table

1. On the **File** menu, point to **Open**, then click **File**.
2. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab07\Starter** folder, click **Drop Indexes on FactInternetSales.sql**, and then click **Open**.
3. Click **Execute** to drop the existing indexes.
4. On the **File** menu, point to **Open**, then click **File**.
5. Click **Create Columnstore Index on FactInternetSales.sql**, and then click **Open**.
6. Click **Execute** to drop the existing indexes.
7. Switch back to the **Query FactInternetSales.sql** tab, and then click **Execute** to rerun the query.

8. On the **Execution plan** tab, scroll to the end to view the execution plan that is used for the query. Examine the icons from right to left, noting the indexes that were used. Note that the new columnstore index is used.
9. What, if any, are the disk space and query performance improvements?
10. Close all the query windows without saving changes.

Results: After completing this exercise, you will have greatly reduced the disk space taken up by the FactInternetSales table, and improved the performance of analytical queries against the table.

Optional Exercise 3: Create a Memory Optimized Columnstore Table

▶ Task 1: Use the Memory Optimization Advisor

1. In SQL Server Management Studio, in **Object Explorer**, expand **Databases**, expand **AdventureWorksDW**, and then expand **Tables**.
2. Right-click **dbo.FactInternetSales**, and then click **Memory Optimization Advisor**.
3. In the **Table Memory Optimization Advisor** window, on the **Introduction** page, click **Next**.
4. On the **Migration validation** page, scroll down, looking at the descriptions against any row in the table without a green tick.
5. Note that the advisor requires that the foreign key relationships are removed before it can continue.
6. Note that the advisor also requires the removal of the clustered columnstore index.
7. Click **Cancel**.

▶ Task 2: Enable the Memory Optimization Advisor to Create a Memory Optimized FactInternetSales Table

1. On the **File** menu, point to **Open**, then click **File**.
2. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab07\Starter** folder, click **Drop Columnstore Indexes on FactInternetSales.sql**, and then click **Open**.
3. Click **Execute** to drop the existing columnstore index and foreign keys.
4. In Object Explorer, right-click **dbo.FactInternetSales**, and then click **Memory Optimization Advisor**.
5. In the **Table Memory Optimization Advisor** window, on the **Introduction** page, click **Next**.
6. On the **Migration validation** page, click **Next**.
7. On the **Memory Optimization Warnings** page, note the warnings and click **Next**.
8. On the **Migration options** page, select the **Also copy table data to the new memory optimized table** check box, and then click **Next**.
9. On the **Index creation** page, in the column list, select the **SalesOrderNumber** and **SalesOrderLineNumber** check boxes, and then click **Next**.
10. On the **Summary** page, click **Script**.
11. Inspect the generated Transact-SQL. Note the code to create the memory optimized filegroup and the renaming of the existing table.

Note: The Memory Optimization Advisor won't suggest columnstore indexes as they are not applicable in all situations. Therefore, you have to add these manually.

12. Locate the following lines of Transact-SQL:

```
)WITH ( BUCKET_COUNT = 2097152)
)WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA )
```

13. Add the following Transact-SQL between the two WITH statements:

```
,INDEX CCI_OnlineFactInternetSales CLUSTERED COLUMNSTORE
```

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

▶ Task 3: Examine the Performance of the Memory Optimized Table

1. On the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab07\Starter** folder, click **Query FactInternetSales.sql**, and then click **Open**.
3. Highlight the Transact-SQL under the Step 1 comment, and then on the toolbar, click **Execute**.
4. Note that no disk space is used, as this table is now memory optimized.
5. On the **Query** menu, click **Include Actual Execution Plan**.
6. Highlight the code after the Step 2 comment, click **Execute**, and then view the query results.
7. On the **Execution plan** tab, view the execution plan for the query, and note the indexes that were used. Position the cursor on the icons to examine their details.
8. Close SQL Server Management Studio without saving changes.

Results: After completing this exercise, you will have created a memory optimized version of the **FactInternetSales** disk based table, using the Memory Optimization Advisor.

Optional Lab 3: Memory-optimized tables

Scenario

You are planning to optimize some database workloads by using the in-memory database capabilities of SQL Server. You will create memory-optimized tables and natively compiled stored procedures to optimize OLTP workloads.

Objectives

After completing this lab, you will be able to:

- Create a memory-optimized table.
- Create a natively compiled stored procedure.

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Using Memory-Optimized Tables

Scenario

The Adventure Works website, through which customers can order goods, uses the InternetSales database. The database already includes tables for sales transactions, customers, and payment types. You need to add a table to support shopping cart functionality. The shopping cart table will experience a high volume of concurrent transactions, so, to maximize performance, you want to implement it as a memory-optimized table.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Add a Filegroup for Memory-Optimized Data

3. Create a Memory-Optimized Table

▶ Task 1: Prepare the Lab Environment

1. Ensure that the MIA-DC and MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab12\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. When you are prompted, click **Yes** to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: Add a Filegroup for Memory-Optimized Data

1. Add a filegroup for memory-optimized data to the InternetSales database.
2. Add a file for memory-optimized data to the InternetSales database. You should store the file in the filegroup that you created in the previous step.

▶ Task 3: Create a Memory-Optimized Table

1. Create a memory-optimized table named **ShoppingCart** in the InternetSales database, with the durability option set to SCHEMA_AND_DATA.
2. The table should include the following columns:
   - **SessionID**: integer
   - **TimeAdded**: datetime
   - **CustomerKey**: integer
   - **ProductKey**: integer
   - **Quantity**: integer
3. The table should include a composite primary key nonclustered index on the **SessionID** and **ProductKey** columns.
4. To test the table, insert the following rows, and then write and execute a SELECT statement to return all of the rows.

SessionID	TimeAdded	CustomerKey	ProductKey	Quantity
1	<*Time*>	2	3	1
1	<*Time*>	2	4	1

For <*Time*>, use whatever the current time is.

Results: After completing this exercise, you should have created a memory-optimized table and a natively compiled stored procedure in a database with a filegroup for memory-optimized data.

Exercise 2: Using Natively Compiled Stored Procedures

Scenario

The Adventure Works website now includes a memory-optimized table. You want to create a natively compiled stored procedure to take full advantage of the performance benefits of in-memory tables.

The main tasks for this exercise are as follows:

1. Create Natively Compiled Stored Procedures

▶ Task 1: Create Natively Compiled Stored Procedures

1. Create a natively compiled stored procedure named **AddItemToCart**. The stored procedure should include a parameter for each column in the **ShoppingCart** table, and should insert a row into the **ShoppingCart** table by using a SNAPSHOT isolation transaction.
2. Create a natively compiled stored procedure named **DeleteItemFromCart**. The stored procedure should include **SessionID** and **ProductKey** parameters, and should delete matching rows from the **ShoppingCart** table by using a SNAPSHOT isolation transaction.
3. Create a natively compiled stored procedure named **EmptyCart**. The stored procedure should include **SessionID** parameters, and should delete matching rows from the **ShoppingCart** table by using a SNAPSHOT isolation transaction.

4. To test the **AddItemToCart** procedure, write and execute a Transact-SQL statement that calls **AddItemToCart** to add the following items, and then write and execute a SELECT statement to return all of the rows in the **ShoppingCart** table.

SessionID	TimeAdded	CustomerKey	ProductKey	Quantity
1	<*Time*>	2	3	1
1	<*Time*>	2	4	1
3	<*Time*>	2	3	1
3	<*Time*>	2	4	1

   For <*Time*>, use whatever the current time is.

5. To test the **DeleteItemFromCart** procedure, write and execute a Transact-SQL statement that calls **DeleteItemFromCart** to delete any items where **SessionID** is equal to 3 and the product key is equal to 4, and then write and execute a SELECT statement to return all of the rows in the **ShoppingCart** table.

6. To test the **EmptyCart** procedure, write and execute a Transact-SQL statement that calls **EmptyCart** to delete any items where **SessionID** is equal to 3, and then write and execute a SELECT statement to return all of the rows in the **ShoppingCart** table.

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

Results: After completing this exercise, you should have created a natively compiled stored procedure.

Optional Lab 3 Answer Key: Memory-optimized tables

Exercise 1: Using Memory-Optimized Tables

▶ Task 1: Prepare the Lab Environment

1. Ensure that the MIA-DC and MIA-SQL virtual machines are both running, and then log on to MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab12\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes** to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: Add a Filegroup for Memory-Optimized Data

1. Start SQL Server Management Studio, and then connect to the **MIA-SQL** database engine instance by using Windows authentication.
2. In SQL Server Management Studio, in Object Explorer, expand **Databases**.
3. Right-click the **InternetSales** database, and then click **Properties**.
4. In the **Database Properties - InternetSales** dialog box, on the **Filegroups** page, in the **MEMORY OPTIMIZED DATA** section, click **Add Filegroup**.
5. In the **Name** box, type **MemFG**, and then press Enter.
6. In the **Database Properties - InternetSales** dialog box, on the **Files** page, click **Add**.
7. In the **Logical Name** column, type **InternetSales_MemData**, and then press Enter.
8. In the **File Type** column, select **FILESTREAM Data**, and then ensure that **MemFG** is automatically selected in the **Filegroup** column.
9. In the **Database Properties - InternetSales** dialog box, click **OK**.

▶ Task 3: Create a Memory-Optimized Table

1. In SQL Server Management Studio, click **New Query**, and then type the following Transact-SQL code.

```
USE InternetSales
GO
CREATE TABLE dbo.ShoppingCart
(SessionID INT NOT NULL,
TimeAdded DATETIME NOT NULL,
CustomerKey INT NOT NULL,
ProductKey INT NOT NULL,
Quantity INT NOT NULL
PRIMARY KEY NONCLUSTERED (SessionID, ProductKey))
WITH  (MEMORY_OPTIMIZED = ON,  DURABILITY = SCHEMA_AND_DATA);
```

2. On the toolbar, click **Execute** to create the table.

3. Click **New Query**, and then type the following Transact-SQL code.

```
USE InternetSales
GO
INSERT INTO dbo.ShoppingCart (SessionID, TimeAdded, CustomerKey, ProductKey,
Quantity)
VALUES (1, GETDATE(), 2, 3, 1);
INSERT INTO dbo.ShoppingCart (SessionID, TimeAdded, CustomerKey, ProductKey,
Quantity)
VALUES (1, GETDATE(), 2, 4, 1);
SELECT * FROM dbo.ShoppingCart;
```

4. On the toolbar, click **Execute** to test the table.

Results: After completing this exercise, you should have created a memory-optimized table and a natively compiled stored procedure in a database with a filegroup for memory-optimized data.

Exercise 2: Using Natively Compiled Stored Procedures

▶ Task 1: Create Natively Compiled Stored Procedures

1. In SQL Server Management Studio, click **New Query**, and then enter the following Transact-SQL code.

```
USE InternetSales
GO
CREATE PROCEDURE dbo.AddItemToCart
@SessionID INT, @TimeAdded DATETIME, @CustomerKey INT, @ProductKey INT, @Quantity INT
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS
BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = 'us_english')
INSERT INTO dbo.ShoppingCart (SessionID, TimeAdded, CustomerKey, ProductKey,
Quantity)
VALUES (@SessionID, @TimeAdded, @CustomerKey, @ProductKey, @Quantity)
END
GO
```

2. On the toolbar, click **Execute** to create the stored procedure.

3. In SQL Server Management Studio, click **New Query**, and then enter the following Transact-SQL code.

```
USE InternetSales
GO
CREATE PROCEDURE dbo.DeleteItemFromCart
@SessionID INT, @ProductKey INT
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS
BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = 'us_english')
DELETE FROM dbo.ShoppingCart
WHERE SessionID = @SessionID
AND ProductKey = @ProductKey
END
GO
```

4. On the toolbar, click **Execute** to create the stored procedure.

5. In SQL Server Management Studio, click **New Query**, and then enter the following Transact-SQL code.

```
USE InternetSales
GO
CREATE PROCEDURE dbo.EmptyCart
@SessionID INT
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS
BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = 'us_english')
DELETE FROM dbo.ShoppingCart
WHERE SessionID = @SessionID
END
GO
```

6. On the toolbar, click **Execute** to create the stored procedure.

7. Click **New Query**, and then enter the following Transact-SQL code.

```
--Step 1 - Use the InternetSales database
USE InternetSales
GO
--Step 2 - Add items to cart
DECLARE @now DATETIME = GETDATE();
EXEC dbo.AddItemToCart
   @SessionID = 3,
   @TimeAdded = @now,
   @CustomerKey = 2,
   @ProductKey = 3,
   @Quantity = 1;
EXEC dbo.AddItemToCart
   @SessionID = 3,
   @TimeAdded = @now,
   @CustomerKey = 2,
   @ProductKey = 4,
   @Quantity = 1;
--Step 3 - Select items in cart
SELECT * FROM dbo.ShoppingCart;
--Step 4 - Delete item from cart
EXEC dbo.DeleteItemFromCart
   @SessionID = 3,
   @ProductKey = 4;
--Step 5 - Select items in cart
SELECT * FROM dbo.ShoppingCart;
--Step 6 - Empty cart
EXEC dbo.EmptyCart
   @SessionID = 3;
--Step 7 - Select items in cart
SELECT * FROM dbo.ShoppingCart;
```

8. Select the code under **Step 1 - Use the InternetSales database**, and then click **Execute**.

9. Select the code under **Step 2 - Add items to cart**, and then click **Execute**.

10. Select the code under **Step 3 - Select items in cart**, and then click **Execute**.

11. Select the code under **Step 4 - Delete item from cart**, and then click **Execute**.

12. Select the code under **Step 5 - Select items in cart**, and then click **Execute**.

13. Select the code under **Step 6 - Empty cart**, and then click **Execute**.

14. Select the code under **Step 7 - Select items in cart**, and then click **Execute**.

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

Results: After completing this exercise, you should have created a natively compiled stored procedure.

Lab 4: XML Data in SQL Server

Scenario

A new developer in your organization has discovered that SQL Server can store XML directly. He is keen to use this mechanism extensively. In this lab, you will decide on the appropriate usage of XML in the documented application.

You also have an upcoming project that will require the use of XML data in SQL Server. No members of your current team have experience working with XML data in SQL Server. You need to learn how to process XML data within SQL Server and you have been given some sample queries to assist with this learning. Finally, you will use what you have learned to write a stored procedure for the marketing system that returns XML data.

Objectives

After completing this lab, you will be able to:

- Determine appropriate use cases for storing XML in SQL Server.
- Test XML storage in variables.
- Retrieve information about XML schema collections.
- Query SQL Server data as XML.
- Write a stored procedure that returns XML.

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Determining When to Use XML

Scenario

In this exercise, you should read the list of scenarios that your new developer has provided. Determine which are appropriate for XML storage in SQL Server, and which are not. Write "Yes" or "No" next to each scenario.

Scenarios

Scenario Requirements
Existing XML data that is stored, but not processed.
Storing attributes for a customer.
Relational data that is being passed through a system, but not processed in it.
Storing attributes that are nested (that is, attributes stored within attributes).

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Review the List of Scenario Requirements

▶ 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\Lab14\Starter** folder as Administrator.

▶ Task 2: Review the List of Scenario Requirements

1. Review the list of requirements.
2. For each requirement, determine whether it is suitable for XML storage.

Results: After completing this exercise, you will have determined the appropriate use cases for XML storage.

Exercise 2: Testing XML Data Storage in Variables

Scenario

In this exercise, you will explore how XML data is stored in variables. From a set of sample XML queries, you will review the effect of executing each query.

The main tasks for this exercise are as follows:

1. Review, Execute, and Review the Results of the XML Queries

▶ Task 1: Review, Execute, and Review the Results of the XML Queries

1. Open SQL Server Management Studio and connect to the MIA-SQL instance of SQL Server using Windows authentication.
2. Open **D:\Labfiles\Lab14\Starter\InvestigateStorage.sql**.
3. For each query in the file, review the code, execute the code, and determine how the output results relate to the queries.
4. Close InvestigateStorage.sql without saving any changes.
5. Leave SSMS open for the next exercise.

Results: After this exercise, you will have seen how XML data is stored in variables.

Exercise 3: Using XML Schemas

Scenario

For some of the XML processing that you will perform in your upcoming project, you need to validate XML data by using XML schemas. In SQL Server, XML schemas are stored in XML schema collections. You need to investigate how these schemas are used. You have been given a set of sample queries to assist with this. In this exercise, you will review the effect of executing these queries.

The main tasks for this exercise are as follows:

1. Review, Execute, and Review the Results of the XML Queries

▶ Task 1: Review, Execute, and Review the Results of the XML Queries

1. Open **D:\Labfiles\Lab14\Starter\XMLSchema.sql**.
2. For each query in the file, review the code, execute the code, and determine how the output results relate to the queries.
3. Leave SSMS open for the next exercise.

Results: After this exercise, you will have seen how to create XML schema collections.

Exercise 4: Using FOR XML Queries

Scenario

For some of the XML processing that you must perform in your upcoming project, you will need to return XML data. You should investigate how to do this. You have been given a set of sample queries to assist with this. In this exercise, you will review the effect of executing these queries.

The main tasks for this exercise are as follows:

1. Review, Execute, and Review the Results of the FOR XML Queries

▶ Task 1: Review, Execute, and Review the Results of the FOR XML Queries

1. Open **D:\Labfiles\Lab14\Starter\XMLQuery.sql**.
2. For each query in the file, review the code, execute the code, and determine how the output results relate to the queries.
3. Leave SSMS open for the next exercise.

Results: After this exercise, you will have seen how to use FOR XML.

Exercise 5: Creating a Stored Procedure to Return XML

Scenario

A new web service is being added to the marketing system. In this exercise, you need to create a stored procedure that will query data from a table and return it as an XML value.

Supporting Documentation

Stored Procedure Specifications

Stored Procedure	**Production.GetAvailableModelsAsXML**
Input Parameters:	None.
Output Parameters:	None.
Returned Rows:	One XML document with attribute-centric XML. Root element is AvailableModels. Row element is AvailableModel. Row contains ProductID, ProductName, ListPrice, Color and SellStartDate (from Marketing.Product) and ProductModelID and ProductModel (from Marketing.ProductModel) for rows where there is a SellStartDate but not yet a SellEndDate.
Output Order:	Rows within the XML should be in order of SellStartDate ascending, and then ProductName ascending. That is, sort by SellStartDate first, and then ProductName within SellStartDate.

Stored Procedure	**Sales.UpdateSalesTerritoriesByXML**
Input Parameters:	@SalespersonMods xml.
Output Parameters:	None.
Returned Rows:	None.
Actions:	Update the SalesTerritoryID column in the Sales.Salesperson table, based on the SalesTerritoryID values extracted from the input parameter.

Incoming XML Object Format

This is an example of the incoming XML:

This is an example of the incoming XML:

Incoming XML Object Format

```
<SalespersonMods>
     <SalespersonMod BusinessEntityID="274">
           <Mods>
               <Mod SalesTerritoryID="3"/>
           </Mods>
     </SalespersonMod>
      <SalespersonMod BusinessEntityID="278">
           <Mods>
                <Mod SalesTerritoryID="4"/>
           </Mods>
     </SalespersonMod>
  </SalespersonMods>
```

The main tasks for this exercise are as follows:

1. Review the Requirements
2. Create a Stored Procedure to Retrieve Available Models
3. Test the Stored Procedure
4. If Time Permits: Create a Stored Procedure to Update the Sales Territories Table

▶ Task 1: Review the Requirements

- Review the stored procedure specification for **WebStock.GetAvailableModelsAsXML** in the exercise scenario.

▶ Task 2: Create a Stored Procedure to Retrieve Available Models

- Create and implement the **Production.GetAvailableModelsAsXML** stored procedure based on the specifications that are provided.

▶ Task 3: Test the Stored Procedure

1. Test the stored procedure by executing it.
2. Review the returned data.

▶ Task 4: If Time Permits: Create a Stored Procedure to Update the Sales Territories Table

1. If time permits, implement the Sales.UpdateSalesTerritoriesByXML stored procedure.
2. Test the created stored procedure with the example incoming XML.
3. Close SSMS without saving any changes.

Results: After this exercise, you will have a new stored procedure that returns XML in the AdventureWorks database.

Lab 4 Answer Key: XML Data in SQLServer

Exercise 1: Determining When to Use XML

▶ 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\Lab14\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 List of Scenario Requirements

1. Review the list of requirements.
2. The following table shows which requirements are suitable for XML storage:

Requirements	Should be implemented	Reason
Existing XML data that is stored, but not processed.	Yes	Data is already XML and does not need to be processed.
Storing attributes for a customer.	No	Should be database columns.
Relational data that is being passed through a system, but not processed within it.	Perhaps	Only if the data is being sent and received as XML.
Storing attributes that are nested (that is, attributes stored within attributes).	Yes	Not standard relational data.

Results: After completing this exercise, you will have determined the appropriate use cases for XML storage.

Exercise 2: Testing XML Data Storage in Variables

▶ Task 1: Review, Execute, and Review the Results of the XML Queries

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**, and then click **Connect**.
3. On the **File** menu, point to **Open**, and then click **File**.
4. In the **Open File** dialog box, navigate to **D:\Labfiles\Lab14\Starter**, click **InvestigateStorage.sql**, and then click **Open**.
5. Select the following code, and then on the toolbar, click **Execute**:

```
USE tempdb;
GO
```

6. Select the code under **Step 14.1**, and then click **Execute**.
7. Select the code under **Step 14.2**, and then click **Execute**. Note that one row is inserted.
8. Select the code under **Step 14.3**, and then click **Execute**. Note that one row is inserted.
9. Select the code under **Step 14.4**, and then click **Execute**. Note that one row is inserted.
10. Select the code under **Step 14.5**, and then click **Execute**. Note that one row is inserted.
11. Select the code under **Step 14.6**, and then click **Execute**. Note that one row is inserted.
12. Select the code under **Step 14.7**, and then click **Execute**. Note that one row is inserted.
13. Select the code under **Step 14.8**, and then click **Execute**. Note that the command fails due to incorrect XML format.
14. Select the code under **Step 14.9**, and then click **Execute**.
15. On the **File** menu, click **Close**.
16. Leave SSMS open for the next exercise.

Results: After this exercise, you will have seen how XML data is stored in variables.

Exercise 3: Using XML Schemas

▶ Task 1: Review, Execute, and Review the Results of the XML Queries

1. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, navigate to **D:\Labfiles\Lab14\Starter**, click **XMLSchema.sql**, and then click **Open**.
3. Select the following code, and then on the toolbar, click **Execute**:

```
USE tempdb;
GO
```

4. Select the code under **Step 14.10**, and then click **Execute**.

5. Select the code under **Step 14.11**, and then click **Execute**. Note there are ten rows showing how the individual components of the XML schema collection are stored. Note the values in the **kind_desc** column.
6. On the **File** menu, click **Close**.
7. Leave SSMS open for the next exercise.

Results: After this exercise, you will have seen how to create XML schema collections.

Exercise 4: Using FOR XML Queries

▶ Task 1: Review, Execute, and Review the Results of the FOR XML Queries

1. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, navigate to **D:\Labfiles\Lab14\Starter**, click **XMLQuery.sql**, and then click **Open**.
3. Select the code under **Step 14.21**, and then click **Execute**. Note the element name is based on the table name.
4. Select the code under **Step 14.22**, and then click **Execute**. Note the element-centric output and the element name is "row".
5. Select the code under **Step 14.23**, and then click **Execute**. Note the products without color show xsi:nil="true" when you view the data by clicking the hyperlink. Other products show the color.
6. Select the code under **Step 14.24**, and then click **Execute**. Note the element name is now Product, based on the alias name of the table.
7. Select the code under **Step 14.25**, and then click **Execute**. Note the inclusion of an XSD schema.
8. Select the code under **Step 14.26**, and then click **Execute**. Note that rows with a value in the **Description** column show that value as XML.
9. Select the code under **Step 14.27**, and then click **Execute**. Note how the output can be constructed with a PATH query. In the output, locate each of the elements in the SELECT clause.
10. Select the code under **Step 14.28**, and then click **Execute**. Note the "AvailableItems" root node.
11. Select the code under **Step 14.29**, and then click **Execute**. Note the "AvailableItem" ElementName.
12. On the **File** menu, click **Close**.
13. Leave SSMS open for the next exercise.

Results: After this exercise, you will have seen how to use FOR XML.

Exercise 5: Creating a Stored Procedure to Return XML

▶ Task 1: Review the Requirements

- Review the stored procedure specification for **WebStock.GetAvailableModelsAsXML** in the exercise scenario.

▶ Task 2: Create a Stored Procedure to Retrieve Available Models

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

```
CREATE PROCEDURE
Production.GetAvailableModelsAsXML
AS BEGIN
SELECT p.ProductID,
p.name as ProductName,
p.ListPrice,
p.Color,
p.SellStartDate,
pm.ProductModelID,
pm.Name as ProductModel
FROM Production.Product AS p
INNER JOIN Production.ProductModel AS pm
ON p.ProductModelID = pm.ProductModelID
WHERE p.SellStartDate IS NOT NULL
AND p.SellEndDate IS NULL
ORDER BY p.SellStartDate, p.Name DESC
FOR XML RAW('AvailableModel'), ROOT('AvailableModels');
END;
GO
```

▶ Task 3: Test the Stored Procedure

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

```
EXEC Production.GetAvailableModelsAsXML;
GO
```

3. In the results pane, click the XML code in the first row.
4. Review the data that the stored procedure returns.

▶ Task 4: If Time Permits: Create a Stored Procedure to Update the Sales Territories Table

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

```
CREATE PROCEDURE Sales.UpdateSalesTerritoriesByXML (@SalespersonMods as xml)
AS BEGIN
              UPDATE  Sales.SalesPerson
              SET TerritoryID = updates.SalesTerritoryID
              FROM    Sales.SalesPerson sp
              INNER JOIN (
                            SELECT
              SalespersonMod.value('@BusinessEntityID','int') AS BusinessEntityID,
SalespersonMod.value('(Mods/Mod/@SalesTerritoryID)[1]','int') AS SalesTerritoryID
                            FROM
@SalespersonMods.nodes('/SalespersonMods/SalespersonMod') as
SalespersonMods(SalespersonMod)) AS updates
              ON sp.BusinessEntityID = updates.BusinessEntityID;
END;
GO
```

3. In Object Explorer, right-click **AdventureWorks**, and then click **New Query**.
4. In the query pane, type the following query to test your stored procedure, and then click **Execute**:

```
DECLARE @xmlTestString nvarchar(2000);
SET @xmlTestString ='
<SalespersonMods>
     <SalespersonMod BusinessEntityID="274">
           <Mods>
               <Mod SalesTerritoryID="3"/>
           </Mods>
     </SalespersonMod>
      <SalespersonMod BusinessEntityID="278">
           <Mods>
                <Mod SalesTerritoryID="4"/>
           </Mods>
     </SalespersonMod>
</SalespersonMods>
)';
DECLARE @testDoc xml;
SET @testDoc = @xmlTestString;
EXEC Sales.UpdateSalesTerritoriesByXML @testDoc;
GO
```

5. Note that two rows in the database have been updated.
6. Close SSMS without saving any changes.

Results: After this exercise, you will have a new stored procedure that returns XML in the AdventureWorks database.

Optional Lab 5: Spatial data in SQL Server

Scenario

Your organization has recently started to acquire spatial data within its databases. The new Marketing database was designed before the company started to implement spatial data. One of the developers has provided a table of the locations where prospects live, called Marketing.ProspectLocation. A second developer has added columns to it for Latitude and Longitude and geocoded the addresses. You will make some changes to the system to help support the need for spatial data.

Objectives

After completing this lab you will be able to:

- Use the GEOMETRY data type.
- Use the GEOGRAPHY data type.
- View the results of spatial queries in SSMS.

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Become Familiar with the geometry Data Type

Scenario

You have decided to learn to write queries using the geometry data type in SQL Server. You will review and execute scripts that demonstrate querying techniques.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Review and Execute Queries

▶ 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\Lab15\Starter** folder, run **Setup.cmd** as Administrator.

▶ Task 2: Review and Execute Queries

1. Start SQL Server Manager Studio and connect to the **MIA-SQL** database instance using Windows authentication.

2. Open the SQL Server solution file **20762_15.ssmssln** located in **D:\Demofiles\Mod15\Starter**.

3. Open the query **51 - Lab Exercise 1.sql**.

4. Execute each section of the script individually and review the results.

Exercise 2: Add Spatial Data to an Existing Table

Scenario

In this lab, you have to modify an existing table, Marketing.ProspectLocation, to replace the existing **Latitude** and **Longitude** columns with a new **Location** column of type **geography**. You must migrate the data to the new **Location** column before you delete the existing **Latitude** and **Longitude** columns.

The main tasks for this exercise are as follows:

1. Add a Location Column to the Marketing.ProspectLocation Table
2. Write Code to Assign Values Based on Existing Latitude and Longitude Columns
3. Drop the Existing Latitude and Longitude Columns

▶ Task 1: Add a Location Column to the Marketing.ProspectLocation Table

- Add a **Location** column to the Marketing.ProspectLocation table in the **MarketDev** database.

▶ Task 2: Write Code to Assign Values Based on Existing Latitude and Longitude Columns

- Write code to assign values to the **Location** column, based on the existing **Latitude** and **Longitude** columns.

▶ Task 3: Drop the Existing Latitude and Longitude Columns

- When you are sure that the new column has the correct data, drop the existing **Latitude** and **Longitude** columns.

Results: After this exercise, you should have replaced the existing **Longitude** and **Latitude** columns with a new **Location** column.

Exercise 3: Find Nearby Locations

Scenario

Your salespeople are keen to visit prospects at their own locations, rather than just talk to them on the phone. To minimize effort, they are keen to simultaneously see other prospects in the same area. You will write a stored procedure that provides details of other prospects in the area. To ensure it performs quickly, you will create a spatial index on the table.

The main tasks for this exercise are as follows:

1. Review the Requirements
2. Create a Spatial Index on the Marketing.ProspectLocation Table
3. Design and Implement the Stored Procedure
4. Test the Procedure

▶ Task 1: Review the Requirements

- Read the Requirements.docx located in the folder D:\Labfiles\Lab15\Starter.

▶ Task 2: Create a Spatial Index on the Marketing.ProspectLocation Table

- Create a spatial index on the **Location** column of the Marketing.ProspectLocation table.

▶ Task 3: Design and Implement the Stored Procedure

- Write a stored procedure that will return the details of all prospects within a given distance of a specified prospect. The stored procedure should have input parameters for **ProspectID** and **Distance in kms** and should output customer details in ascending distance order.

▶ Task 4: Test the Procedure

1. Execute the stored procedure to find all prospects within 50 kms of prospectID 2 and check the results.
2. Close SSMS without saving anything.

Results: After completing this lab, you will have created a spatial index and written a stored procedure that will return the prospects within a given distance from a chosen prospect.

Optional Lab 5 Answer Key: Spatial data in SQL Server

Exercise 1: Become Familiar with the geometry Data Type

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **20762C-MIA-DC** and 2**0762C-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 File Explorer, navigate to the **D:\Labfiles\Lab15\Starter** folder, right-click the **Setup.cmd** file, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish. Press any key to continue.

▶ Task 2: Review and Execute Queries

1. Start SQL Server Manager Studio and connect to the **MIA-SQL** database instance using Windows authentication.
2. On the **File** menu, click **Open**, click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to **D:\Labfiles\Lab15\Starter**, and then double-click **20762_15.ssmssln**.
4. In Solution Explorer, expand **Queries**, and double-click **51 - Lab Exercise 1.sql** to open the query.
5. Highlight the text under the comment **Script 15.1 Draw a square**, and click **Execute**.
6. Click the **Spatial results** tab to see the output of the script.
7. Highlight the text under the comment **Script 15.2 Try an invalid value - note the 6522 error and the wrapped 24306 error message**, and click **Execute**.
8. Review the error message produced.
9. Highlight the text under the comment **Script 15.3 Draw a more complex shape**, and click **Execute**.
10. Click the **Spatial results** tab to see the output of the script.
11. Highlight the text under the comment **Script 15.4 Multiple shapes**, and click **Execute**.
12. Click the **Spatial results** tab to see the output of the script.
13. Highlight the text under the comment **Script 15.5 Intersecting shapes**, and click **Execute**.
14. Click the **Spatial results** tab to see the output of the script.
15. Highlight the text under the comment **Script 15.6 Union of the two shapes**, and click **Execute**.
16. Click the **Spatial results** tab to see the output of the script.
17. Highlight the text under the comment **Script 15.7 Intersection of shapes**, and click **Execute**.
18. Click the **Spatial results** tab to see the output of the script.
19. Highlight the text under the comment **Script 15.8 Draw Australia**, and click **Execute**.
20. Click the **Spatial results** tab to see the output of the script.
21. Highlight the text under the comment **Script 15.9 Draw Australia with a buffer around it**, and click **Execute**.
22. Click the **Spatial results** tab to see the output of the script.

Exercise 2: Add Spatial Data to an Existing Table

▶ Task 1: Add a Location Column to the Marketing.ProspectLocation Table

1. In Object Explorer, expand **MIA-SQL (SQL Server 13.0.1000 - ADVENTUREWORKS\Student)**, expand **Databases**, right-click the **MarketDev** database, and then click **New Query**.
2. In the query pane, type the following query:

```
ALTER TABLE Marketing.ProspectLocation
ADD Location GEOGRAPHY NULL;
GO
```

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

▶ Task 2: Write Code to Assign Values Based on Existing Latitude and Longitude Columns

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

```
UPDATE Marketing.ProspectLocation
SET Location = GEOGRAPHY::STGeomFromText('POINT(' + CAST(Longitude AS varchar(20))+ '
' + CAST(Latitude AS varchar(20))+ ')',4326);
GO
```

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

▶ Task 3: Drop the Existing Latitude and Longitude Columns

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

```
ALTER TABLE Marketing.ProspectLocation
DROP COLUMN Latitude;
GO
ALTER TABLE Marketing.ProspectLocation
DROP COLUMN Longitude;
GO
```

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

Results: After this exercise, you should have replaced the existing **Longitude** and **Latitude** columns with a new **Location** column.

Exercise 3: Find Nearby Locations

▶ Task 1: Review the Requirements

1. In File Explorer, navigate to the folder **D:\Labfiles\Lab15\Starter** and open the file **Requirements.docx**.
2. Read the requirements document to familiarize yourself with the requirements.

▶ Task 2: Create a Spatial Index on the Marketing.ProspectLocation Table

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

```
CREATE SPATIAL INDEX IX_ProspectLocation_Location
ON Marketing.ProspectLocation(Location);
GO
```

▶ Task 3: Design and Implement the Stored Procedure

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

```
CREATE PROCEDURE Marketing.GetNearbyProspects
( @ProspectID int,
  @DistanceInKms int
)
AS BEGIN
  DECLARE @LocationToTest GEOGRAPHY;
  SET @LocationToTest = (SELECT pl.Location
                         FROM Marketing.ProspectLocation AS pl
                         WHERE pl.ProspectID = @ProspectID);
  SELECT pl.Location.STDistance(@LocationToTest) / 1000 AS Distance,
         p.ProspectID,
         p.LastName,
         p.FirstName,
         p.WorkPhoneNumber,
         p.CellPhoneNumber,
         pl.AddressLine1,
         pl.AddressLine2,
         pl.City
  FROM Marketing.Prospect AS p
  INNER JOIN Marketing.ProspectLocation AS pl
  ON p.ProspectID = pl.ProspectID
  WHERE pl.Location.STDistance(@LocationToTest) < (@DistanceInKms * 1000)
  AND p.ProspectID <> @ProspectID
  ORDER BY Distance;
END;
```

3. GO

▶ Task 4: Test the Procedure

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

```
EXEC Marketing.GetNearbyProspects 2,50;
GO
```

3. Close SSMS without saving anything.

Results: After completing this lab, you will have created a spatial index and written a stored procedure that will return the prospects within a given distance from a chosen prospect.

Lab 6: BLOBs and Text Documents in SQL Server

Scenario

You manager has asked you to evaluate and optimize the performance of queries against the **LargePhoto** column in the **Production.ProductPhotos** table. You have decided that, because BLOBs in this column are often larger than 1 MB, it will be advantageous to create a FILESTREAM column and move the existing data into the new column.

You have also been asked to create a FileTable, with a corresponding shared folder, so users can store documents by using Word and other desktop applications. These files will be accessible through the file share and database queries.

Finally, you have also been asked to create a full-text index on the **Description** column in the **Production.ProductDescriptions** table so that generation term queries and thesaurus queries can be used.

Objectives

At the end of this lab, you will be able to:

- Enable FILESTREAM and move BLOB data into a FILESTREAM column.
- Enable FileTables and create a FileTable.
- Create and query a full-text index.

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Enabling and Using FILESTREAM Columns

Scenario

Having decided to move BLOB data into a FILESTREAM column, you will now implement that strategy.

The main tasks for this exercise are as follows:

1. Prepare the Environment
2. Enable FILESTREAM for the SQL Server Instance
3. Enable FILESTREAM for the Database
4. Create a FILESTREAM Column
5. Move Data into the FILESTREAM Column

▶ Task 1: Prepare the Environment

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\Lab16\Starter** folder as Administrator.

▶ Task 2: Enable FILESTREAM for the SQL Server Instance

1. Using SQL Server Configuration Manager, enable **FILESTREAM** for the SQL Server (MSSQLSSERVER) instance. Enable **FILESTREAM** for file I/O access and give all remote clients access to FILESTREAM data.
2. Restart the SQL Server (MSSQLSERVER) service.
3. Open the project file **D:\Labfiles\Lab16\Starter\Project\Project.ssmssln** and the T-SQL script **Lab Exercise 1.sql**. Ensure that you are connected to the **AdventureWorks** database.
4. Write and execute a script that uses the sp_configure stored procedure to set the **FILESTREAM** access level to 2.

▶ Task 3: Enable FILESTREAM for the Database

1. In SQL Server Management Studio, write a query to add a FILESTREAM filegroup called **AdWorksFilestreamGroup** to the **AdventureWorks2016** database.
2. Write a query to add a file named **D:\FilestreamData** to the **AdventureWorks** database.

▶ Task 4: Create a FILESTREAM Column

1. In SQL Server Management Studio, write a query to add a new UNIQUEIDENTIFIER, non-nullable row GUID column called **PhotoGuid** to the **Production.ProductPhoto** table.
2. Write a query to enable **FILESTREAM** for the **Product.ProductPhoto** table, and add BLOBs to the **AdworksFilestreamGroup**.
3. Write a query to add a new column called **NewLargePhoto** to the **Production.ProductPhoto** table. Ensure the new column has FILESTREAM enabled and is a nullable varbinary(max) column.

▶ Task 5: Move Data into the FILESTREAM Column

1. In SQL Server Management Studio, write a query to copy all data from the **LargePhoto** column into the **NewLargePhoto** column.
2. Write a query to drop the **LargePhoto** column from the **Production.ProductPhoto** table.
3. Write a query that used the sp_rename stored procedure to change the name of the **NewLargePhoto** column **to LargePhoto**.
4. Close the Lab Exercise 1.sql script.

Results: At the end of this exercise, you will have:

Enabled FILESTREAM on the SQL Server instance.

Enabled FILESTREAM on a database.

Moved data into the FILESTREAM column.

Exercise 2: Enabling and Using FileTables

Scenario

You have decided to allow users to be able to store files in the database, and will now implement that strategy with FileTables.

The main tasks for this exercise are as follows:

1. Enable Nontransactional Access
2. Create a FileTable
3. Add a File to the FileTable

▶ Task 1: Enable Nontransactional Access

1. In SQL Server Management Studio, open the T-SQL script **Lab Exercise 2.sql**.
2. Write a query that uses the sys.database_filestream_options system view to display whether nontransacted access is enabled for each database in the instance.
3. Write a query that enables nontransacted access for the **AdventureWorks2016** database. Set the transacted access level to full and the directory name to **"FileTablesDirectory"**.

▶ Task 2: Create a FileTable

1. In SQL Server Management Studio, write a query to create a new FileTable in the **AdventureWorks2016 database**. Name the FileTable **"DocumentStore"** and use a FileTable directory named **"DocumentStore"**.
2. Write a query that uses the sys.database_filestream_options system view to display whether nontransacted access is enabled for each database in the instance.
3. Write a query that uses the sys.filetables system view to list the FileTables in the **AdventureWorks** database.

▶ Task 3: Add a File to the FileTable

1. In SQL Server Management Studio, write a query that uses the **FileTableRootPath()** function to find the path to the file share for the **DocumentStore** filetable.
2. Copy and paste the path you determined into the address bar of a new File Explorer window.
3. Create a new text document called **DocumentStoreTest** in the file table shared folder.
4. In SQL Management Studio, write a query that displays all rows in the DocumentStore FileTable.

Results: At the end of this exercise, you will have:

Enabled nontransactional access.

Created a FileTable.

Added a file to the FileTable.

Exercise 3: Using a Full-Text Index

Scenario

You will now create a full-text index on the **Description** column in the **Production.ProductDescriptions** table so that generation term queries and thesaurus queries can be used.

The main tasks for this exercise are as follows:

1. Create a Full-Text Index
2. Using a Full-Text Index

▶ Task 1: Create a Full-Text Index

1. In SQL Server Management Studio, open the T-SQL script **Lab Exercise 3.sql**.
2. Execute the first SELECT query in the Lab Exercise 2.sql script file, which lists the tables that have a full- text index in the **Adventure Works2016** database.
3. Write a query that creates a new full-text catalog in the **Adventure Works2016** database with the name **ProductFullTextCatalog**.
4. Write a query that creates a new unique index called **ui_ProductDescriptionID** and indexes the **ProductDescriptionID** column in the **Production.ProductDescription** table.
5. Write a query that creates a new full-text index on the **Description** column of the **Production.ProductDescription** table. Use the ui_ProductDescription unique index and the ProductFullTextCatalog.

▶ Task 2: Using a Full-Text Index

1. In SQL Server Management Studio, write a script that executes a simple term query against the Description column in the **Production.ProductDescription** table. Locate rows that contain the word "Bike". Make a note of the number of rows returned.
2. Write a script that executes a generation term query against the **Description** column in the **Production.ProductDescription** table. Locate rows that contain the word "Bike". Make a note of the number of rows returned.
3. Write a script that returns rows from the previous generation term query but not terms from the previous simple terms query. Examine the Description text for these results.
4. Close Microsoft SQL Server Management Studio, without saving any changes.

Results: At the end of this exercise, you will have created a full-text index.

Lab 6 Answer Key: BLOBs and TextDocuments in SQL Server

Exercise 1: Enabling and Using FILESTREAM Columns

▶ Task 1: Prepare the Environment

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\Lab16\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**.
4. At the command prompt, if the **Do you want to continue this operation?** message appears, type **y**, and press Enter.
5. Wait for the script to finish, and then press Enter.

▶ Task 2: Enable FILESTREAM for the SQL Server Instance

1. On the Start menu, type **SQL Server 2017 Configuration Manager**, and then click **SQL Server 2017 Configuration Manager**.
2. In the **User Account Control** dialog box, click **Yes**.
3. In the hierarchy on the left, click **SQL Server Services**.
4. In the list of services, right-click **SQL Server (MSSQLSERVER)**, and then click **Properties**.
5. In the **SQL Server (MSSQLServer) Properties** dialog box, on the **FILESTREAM** tab, ensure the **Enable FILESTREAM for Transact-SQL access** check box is selected.
6. Ensure the **Enable FILESTREAM for file I/O access** check box is selected.
7. Ensure the **Allow remote clients access to FILESTREAM data** check box is selected, and then click **OK**.
8. In the list of services, right-click **SQL Server (MSSQLSERVER)**, and then click **Restart**.
9. When the service has restarted, close SQL Server Configuration Manager.
10. On the taskbar, click **Microsoft SQL Server Management Studio**.
11. In the **Connect to Server** dialog box, in the **Server name** box, type **MIA-SQL**, and then click **Options**.
12. On the **Connection Properties** tab, in the **Connect to database** list, click **<Browse server...>**.
13. In the **Browse for Databases** dialog box, click **Yes**.
14. In the **Browse Server for Databases** dialog box, under **User Databases**, click **AdventureWorks2016**, and then click **OK**.

15. In the **Connect to Server** dialog box, on the **Login** tab, in the **Authentication** list, click **Windows Authentication**, and then click **Connect**.
16. On the **File** menu, point to **Open** and click **Project/Solution**.
17. In the **Open Project** dialog box, navigate to **D:\Labfiles\Lab16\Starter\Project**, click **Project.ssmssln**, and then click **Open**.
18. In Solution Explorer, expand **Queries**, and double-click the **Lab Exercise 1.sql** query.
19. In the query pane, after the **Task 1** description, type the following script:

```
EXEC sp_configure
   filestream_access_level, 2;
RECONFIGURE;
GO
```

20. Select the script and click **Execute**.

▶ Task 3: Enable FILESTREAM for the Database

1. In SQL Server Management Studio, in the query pane, after the **Task 2** description, type the following script:

```
ALTER DATABASE AdventureWorks2016
ADD FILEGROUP AdworksFilestreamGroup CONTAINS FILESTREAM;
GO
```

2. Select the query and click **Execute**.
3. Under the previously entered code, type the following script:

```
ALTER DATABASE AdventureWorks2016
ADD FILE (NAME='FilestreamData', FILENAME='D:\FilestreamData')
TO FILEGROUP AdworksFilestreamGroup;
GO
```

4. Select the query and click **Execute**.

▶ Task 4: Create a FILESTREAM Column

1. In SQL Server Management Studio, in the query pane, after the **Task 3** description, type the following script:

```
USE AdventureWorks2016;
 GO
ALTER TABLE Production.ProductPhoto
ADD PhotoGuid UNIQUEIDENTIFIER NOT NULL ROWGUIDCOL UNIQUE DEFAULT newid();
GO
```

2. Select the query and click **Execute**.
3. Under the previously entered code, type the following script:

```
ALTER TABLE Production.ProductPhoto
SET (filestream_on = AdworksFilestreamGroup);
 GO
```

4. Select the query and click **Execute**.

5. Under the previously entered code, type the following script:

```
ALTER TABLE Production.ProductPhoto
ADD NewLargePhoto varbinary(max) FILESTREAM NULL;
 GO
```

6. Select the query and click **Execute**.

▶ Task 5: Move Data into the FILESTREAM Column

1. In SQL Server Management Studio, in the query pane, after the **Task 4** description, type the following script:

```
UPDATE Production.ProductPhoto
SET NewLargePhoto = LargePhoto;
GO
```

2. Select the query and click **Execute**.
3. Under the previously entered code, type the following script:

```
ALTER TABLE Production.ProductPhoto
DROP COLUMN LargePhoto;
 GO
```

4. Select the query and click **Execute**.
5. Under the previously entered code, type the following script:

```
EXEC sp_rename 'Production.ProductPhoto.NewLargePhoto', 'LargePhoto', 'COLUMN';
GO
```

6. Select the query and click **Execute**. Note the caution.
7. On the **File** menu, click **Close**.
8. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes**.

Results: At the end of this exercise, you will have:

Enabled FILESTREAM on the SQL Server instance.

Enabled FILESTREAM on a database.

Moved data into the FILESTREAM column.

Exercise 2: Enabling and Using FileTables

▶ Task 1: Enable Nontransactional Access

1. In SQL Server Management Studio, in Solution Explorer, double-click **Lab Exercise 2.sql**.
2. In the query pane, after the **Task 1** description, type the following script:

```
SELECT DB_NAME(database_id) AS dbname, non_transacted_access,
non_transacted_access_desc
    FROM sys.database_filestream_options;
GO
```

3. Select the query and click **Execute**.
4. Under the previously entered code, type the following script:

```
ALTER DATABASE AdventureWorks2016
SET FILESTREAM ( NON_TRANSACTED_ACCESS = FULL, DIRECTORY_NAME =
N'FileTablesDirectory' );
GO
```

5. Select the query and click **Execute**.

▶ Task 2: Create a FileTable

1. In SQL Server Management Studio, in the query pane, after the **Task 2** description, type the following script:

```
USE AdventureWorks2016;
GO
CREATE TABLE dbo.DocumentStore AS FileTable
WITH (FileTable_Directory = 'DocumentStore');
GO
```

2. Select the query and click **Execute**.
3. Under the previously entered code, type the following script:

```
SELECT DB_NAME(database_id) AS dbname, non_transacted_access,
non_transacted_access_desc
    FROM sys.database_filestream_options;
GO
```

4. Select the query and click **Execute**.
5. Under the previously entered code, type the following script:

```
SELECT * FROM AdventureWorks2016.sys.filetables;
```

6. Select the query and click **Execute**.

▶ Task 3: Add a File to the FileTable

1. In SQL Server Management Studio, in the query pane, after the **Task 3** description, type the following script:

```
SELECT
FileTableRootPath('dbo.DocumentStore');
 GO
```

2. Select the query and click **Execute**.

3. In the **Results** pane, right-click the only result, and then click **Copy**.
4. On the taskbar, right-click **File Explorer**, and then click **File Explorer**.
5. Click the address bar, right-click the address bar, click **Paste**, and then press Enter.
6. In the **DocumentStore** folder, on the **Home** menu, click **New item**, and then click **Text Document**.
7. Type **DocumentStoreTest**, and then press Enter.
8. Switch to SQL Server Management Studio.
9. Under the previously entered code, type the following script:

```
SELECT * FROM dbo.DocumentStore;
```

10. Select the query and click **Execute**.

Results: At the end of this exercise, you will have:

Enabled nontransactional access.

Created a FileTable.

Added a file to the FileTable.

Exercise 3: Using a Full-Text Index

▶ Task 1: Create a Full-Text Index

1. In SQL Server Management Studio, in the Solution Explorer, double-click **Lab Exercise 3.sql**.
2. In the query pane, after the **Task 1** description, highlight and execute the script.
3. Under the previously entered code, type the following script:

```
CREATE FULLTEXT CATALOG ProductFullTextCatalog;
 GO
```

4. Select the query and click **Execute**.
5. Under the previously entered code, type the following script:

```
CREATE UNIQUE INDEX ui_ProductDescriptionID ON
Production.ProductDescription(ProductDescriptionID);
GO
```

6. Select the query and click **Execute**.
7. Under the previously entered code, type the following script:

```
CREATE FULLTEXT INDEX ON Production.ProductDescription
  ( Description Language 1033 )
KEY INDEX ui_ProductDescriptionID
ON ProductFullTextCatalog;
GO
```

8. Select the query and click **Execute**.

▶ Task 2: Using a Full-Text Index

1. In SQL Server Management Studio, in the query pane, after the **Task 2** description, type the following script:

```
SELECT ProductDescriptionID, Description
FROM Production.ProductDescription
WHERE CONTAINS(Description, 'Bike');
```

2. Select the query and click **Execute**. Make a note of the number of rows returned.
3. Under the previously entered code, type the following script:

```
SELECT ProductDescriptionID, Description
FROM Production.ProductDescription
WHERE CONTAINS(Description, 'FORMSOF(INFLECTIONAL, Bike)');
```

4. Select the query and click **Execute**. Make a note of the number of rows returned.
5. Under the previously entered code, type the following script:

```
SELECT ProductDescriptionID, Description
FROM Production.ProductDescription
WHERE CONTAINS(Description, 'FORMSOF(INFLECTIONAL, Bike)') AND NOT
CONTAINS(Description, 'Bike');
```

6. Select the query and click **Execute**. Make a note of the number of rows returned.
7. Close Microsoft SQL Server Management Studio, without saving any changes.

Results: At the end of this exercise, you will have created a full-text index.

Lab 7: Performance and monitoring

Scenario

You are investigating why a new SQL Server instance is running slowly. Users are complaining that their workloads run particularly slowly during peak business hours. To troubleshoot these performance issues, and take informed corrective measures, you decide to establish a baseline for SQL Server performance.

In this lab exercise, you will set up data collection for analyzing workloads during peak business hours, and implement a baseline methodology to collect performance data at frequent intervals. This will enable comparisons to be made with the baseline.

Objectives

After completing this lab, you will be able to:

- Collect and analyze performance data by using Extended Events.
- Implement a methodology to establish a baseline.

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Collecting and Analyzing Data Using Extended Events

Scenario

You have been asked to prepare a reusable Extended Events session to collect and analyze workload.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Set Up an Extended Event Session

3. Execute Workload

4. Analyze Collected Data

▶ 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\Lab18\Starter** folder as Administrator.

▶ Task 2: Set Up an Extended Event Session

1. Create an Extended Events session to capture the **sqlserver.error_reported**, **sqlserver.module_end**, **sqlserver.sp_statement_completed**, and **sqlserver.sql_batch_completed** events with a **ring_buffer** target. In the **D:\Labfiles\Lab18\Starter\20762** folder, the **SetupExtendedEvent.sql** file has a possible solution script.

2. Use Watch Live Data for the Extended Events session.

▶ Task 3: Execute Workload

1. In the **D:\Labfiles\Lab18\Starter** folder, in the **RunWorkload.cmd** file, run the workload multiple times to generate event data for the **Extended Event** session.
2. In the **AnalyzeSQLEE** session live data window, stop the feed data, and then add the **duration**, **query_hash**, and **statement** columns to the view.

▶ Task 4: Analyze Collected Data

1. In the AnalyzeSQLEE Extended Events live data window, group the data on **query_hash** data, and then aggregate the data on **average of duration**. Sort the data in descending order of duration so that statements that take the highest average time are at the top.
2. Review the data in one of the query hash rows.

   Drop the AnalyzeSQLEE Extended Events session.

Results: At the end of this lab, you will be able to:

Set up an Extended Events session that collects performance data for a workload.

Analyze the data.

Lab 7 Answer Key: Performance and monitoring

Exercise 1: Collecting and Analyzing Data Using Extended Events

▶ 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\Lab18\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**.
4. If you are prompted with the question **Do you want to continue with this operation?**, type **y**, press Enter, and then wait for the script to finish.

▶ Task 2: Set Up an Extended Event Session

1. Start SQL Server Management Studio, and then connect to the **MIA-SQL** database engine instance by using Windows authentication.
2. On the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab18\Starter\20762** folder, click **20762-18.ssmsslnproj**, and then click **Open**.
4. In Solution Explorer, under **Queries**, double-click **SetupExtendedEvent.sql**.
5. Examine the contents of the script, and then click **Execute**.
6. In Object Explorer, expand **Management**, expand **Extended Events**, and then expand **Sessions**.
7. Right-click **AnalyzeSQLEE**, and then click **Watch Live Data**.

▶ Task 3: Execute Workload

1. In File Explorer, in the **D:\Labfiles\Lab18\Starter** folder, right-click **RunWorkload.cmd**, and then click **Run as administrator**.
2. In the **User Account Control** dialog box, click **Yes**.
3. After execution of the workload completes, repeat steps 1 and 2.
4. In SQL Server Management Studio, on the **Extended Events** menu, click **Stop Data Feed**.
5. In the **MIA-SQL - AnalyzeSQLEE: Live Data** query pane, right-click the **name** column heading, and then click **Choose Columns**.
6. In the **Choose Columns** dialog box, under **Available columns**, click **duration**, click **>**, click **query_hash**, click **>**, click **statement**, click **>**, and then click **OK**.

▶ Task 4: Analyze Collected Data

1. In the **MIA-SQL - AnalyzeSQLEE: Live Data** query, right-click the **query_hash** column heading, and then click **Group by this Column**.
2. Right-click the **duration** column heading, point to **Calculate Aggregation,** and then click **AVG**.
3. Right-click the **duration** column heading, and then click **Sort Aggregation Descending**.

4. Expand one of the query hash rows to observe the top statements by duration.
5. In Solution Explorer, double-click **cleanup.sql**.
6. Examine the contents of the script, and then click **Execute** to remove the Extended Event.
7. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this lab, you will be able to:

Set up an Extended Events session that collects performance data for a workload.

Analyze the data.