Lab Manual

T1987, Performance Tuning and Optimizing SQL Databases

Contents

Lab 1: SQL Server Architecture, Scheduling, and Waits

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler, and retailer of cycle products. The owners of the company have decided to start a new direct marketing arm. This has been created as a new company named Proseware Inc. Even though Proseware Inc. has been set up as a separate company, it will receive some IT-related services from Adventure Works and will be provided with a subset of the corporate Adventure Works data. The existing Adventure Works SQL Server platform has been moved to a new server that can support both the existing workload and the workload from the new company.

Objectives

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

- Explore the configuration of database engine components, schedulers, and NUMA.
- Monitor workload pressure on schedulers and observe the life cycle of a thread.
- Monitor and record wait statistics.

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Recording CPU and NUMA Configuration

Scenario

A new instance of SQL Server has been installed by the IT department at Adventure Works. In the first exercise, you need to document CPU and NUMA configuration.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Record CPU Configuration
3. Record CPU-Related Configuration Settings
4. Record NUMA Configuration
5. Record Distribution of Schedulers Across NUMA Nodes

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. Run Setup.cmd in the D:\Labfiles\Lab01\Starter folder as Administrator.

▶ Task 2: Record CPU Configuration

1. Start SQL Server Management Studio, and then open the project file **D:\Labfiles\Lab01\Starter\Project\Project.ssmssln** and the Transact-SQL file **Lab Exercise 01 - CPU and NUMA.sql**.
2. Under the heading for Task 1, write a query to return details of the CPU and hyperthreading configuration of the server that is hosting the MIA-SQL instance. Hint: look for CPU count and hyperthreading ratio values in the output of the **sys.dm_os_sys_info** DMV.

▶ Task 3: Record CPU-Related Configuration Settings

- Edit the query under the heading for Task 2 to return the following additional configuration values:
  - Max degree of parallelism
  - Max worker threads
  - Priority boost

▶ Task 4: Record NUMA Configuration

- Under the heading for Task 3, write a query to return details of the NUMA configuration for this server. Hint: the **sys.dm_os_nodes** DMV will provide the information that you need.

▶ Task 5: Record Distribution of Schedulers Across NUMA Nodes

- Under the heading for Task 4, write a query to return details of how user schedulers are distributed across NUMA nodes for this SQL Server instance. Hint: you will need to join **sys.dm_os_nodes** with **sys.dm_os_schedulers** on **node_id** and **parent_node_id**.

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

Record CPU configuration.

Record NUMA configuration.

Exercise 2: Monitoring Schedulers and User Requests

Scenario

The new instance of SQL Server supports the existing workload and the workload from the new company. In this exercise, you will monitor the scheduling in SQL Server. You will monitor workload pressure on schedulers and the life cycle of threads.

The main tasks for this exercise are as follows:

1. Start the Workload

2. Monitor Workload Pressure on Schedulers

3. Monitor Task Status for User Requests

4. Stop the Workload

▶ Task 1: Start the Workload

- In the D:\Labfiles\Lab01\Starter folder, execute **start_load_exercise_02.ps1** by using Windows PowerShell®.

▶ Task 2: Monitor Workload Pressure on Schedulers

1. In Solution Explorer, open the **Lab Exercise 02 - Monitor Schedulers.sql** query file.
2. Under the heading for Task 2, write a query to return details of the visible online schedulers.
3. How do the column values change as the workload runs?
4. Can you make any deductions about the level of CPU pressure?

▶ Task 3: Monitor Task Status for User Requests

1. Under the heading for Task 3, write a query to return details of active user requests. Hint: to filter the output, only include sessions that have a **session_id** value of greater than 50. Sessions that have an ID of less than 50 are likely to be system sessions.
2. Which wait type are the user requests waiting for?
3. What does the wait type indicate?

▶ Task 4: Stop the Workload

- Highlight the code under the heading for Task 4, and then execute it.

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

Monitor workload pressure on schedulers.

Monitor thread status for user requests.

Exercise 3: Monitoring Waiting Tasks and Recording Wait Statistics

Scenario

The additional workload is causing the new SQL Server instance to respond slowly. Users are occasionally complaining of poor performance and general slowness. In this exercise, you will monitor and record wait statistics.

The main tasks for this exercise are as follows:

1. Clear Wait Statistics
2. Check Current Wait Statistics
3. Start the Workload
4. Monitor Waiting Tasks While the Workload Is Running
5. Record Wait Statistics for Analysis
6. Stop the Workload

▶ Task 1: Clear Wait Statistics

1. In Solution Explorer, open the **Lab Exercise 03 - Waits.sql** query file.
2. Under the heading for Task 1, write a query to clear wait statistics. Hint: review the topic *Viewing Wait Statistics* in Lesson 3 of this module for assistance with this task.

▶ Task 2: Check Current Wait Statistics

- Under the heading for Task 2, write a query to select all rows and columns from the wait statistics DMV.

▶ Task 3: Start the Workload

- In the D:\Labfiles\Lab01\Starter folder, execute **start_load_exercise_03.ps1** by using Windows PowerShell.

▶ Task 4: Monitor Waiting Tasks While the Workload Is Running

- Under the heading for Task 4, write a query to view the waiter list. Hint: to filter the output, only include sessions that have a **session_id** value of greater than 50. Sessions that have an ID that is less than 50 are likely to be system sessions.

Note: Note the wait type(s) for which the tasks are waiting.

▶ Task 5: Record Wait Statistics for Analysis

1. Execute the first query under the heading for Task 5 to capture a snapshot of wait statistics into a temporary table called **#wait_stats_snapshot**.
2. The second query under the heading for Task 5 compares the snapshot that was captured in **#wait_stats_snapshot** with the current wait statistics.
   Amend this query to order the results by the change to **wait_time_ms** between the snapshot and the current wait statistics, descending, and exclude wait types where there is no change.

▶ Task 6: Stop the Workload

- Highlight the code under the heading for Task 6, and then execute it.

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

Monitor the waiting tasks list.

Capture and review wait statistics.

Lab Answer Key 1: SQL Server Architecture, Scheduling,and Waits

Exercise 1: Recording CPU and NUMA Configuration

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. In the D:\Labfiles\Lab01\Starter folder, right-click **Setup.cmd**, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Record CPU Configuration

1. Start SQL Server Management Studio, and then connect to the **MIA-SQL** database engine by using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the Open Project window, open the **D:\Labfiles\Lab01\Starter\Project\Project.ssmssln** project.
4. In Solution Explorer, double-click the **Lab Exercise 01 - CPU and NUMA.sql** query. (If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**, or press Ctrl+Alt+L on the keyboard.)
5. Under the heading for Task 1, type the following:

```
SELECT cpu_count, hyperthread_ratio
FROM sys.dm_os_sys_info;
```

6. Highlight the query that you have typed, and then click **Execute** (or press F5 or Ctrl+E).

▶ Task 3: Record CPU-Related Configuration Settings

1. In SQL Server Management Studio, in the query pane, under the heading for Task 2, edit the query so that it reads as follows:

```
SELECT *
FROM sys.configurations
WHERE name IN
  ('affinity mask',
   'affinity64 mask',
   'cost threshold for parallelism',
   'lightweight pooling',
   'max degree of parallelism',
   'max worker threads',
   'priority boost');
```

2. Highlight the query that you have typed, and then click **Execute**.

▶ Task 4: Record NUMA Configuration

1. In SQL Server Management Studio, in the query pane, under the heading for Task 3, type the following:

```
SELECT * FROM sys.dm_os_nodes;
```

2. Highlight the query that you have typed, and then click **Execute**.

▶ Task 5: Record Distribution of Schedulers Across NUMA Nodes

1. In SQL Server Management Studio, in the query pane, edit the query under the heading for Task 4 so that it reads as follows:

```
SELECT OSS.scheduler_id, OSS.status, OSS.parent_node_id, OSN.node_state_desc
FROM sys.dm_os_schedulers
AS OSS
JOIN sys.dm_os_nodes AS OSN
ON OSS.parent_node_id = OSN.node_id;
```

2. Highlight the query, and then click **Execute**.

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

Record CPU configuration.

Record NUMA configuration.

Exercise 2: Monitoring Schedulers and User Requests

▶ Task 1: Start the Workload

1. Open Windows Explorer, and then navigate to D:\Labfiles\Lab01\Starter.
2. Right-click **start_load_exercise_02.ps1**, and then click **Run with PowerShell**. Leave the script running, and continue with the lab.

▶ Task 2: Monitor Workload Pressure on Schedulers

1. In Solution Explorer, double-click the **Lab Exercise 02 - Monitor Schedulers.sql** query.
2. When the query window opens, in the query pane, type the following query after the Task 2 description:

```
SELECT *
FROM sys.dm_os_schedulers
WHERE status = 'VISIBLE ONLINE';
```

3. Highlight the query, and then click **Execute**.
4. Execute this query several times and notice how the column values change. The value of runnable_tasks_count gives an indication of the length of the runnable queue, and therefore of CPU pressure.

▶ Task 3: Monitor Task Status for User Requests

1. In the query pane, type the following under the heading for Task 3:

```
SELECT *
FROM sys.dm_exec_requests
WHERE session_id > 50;
```

2. Highlight the code that you have typed, and then click **Execute**.
3. The workload sessions will be those with a command of **SELECT** and a non-NULL **sql_handle**. The workload sessions are likely to be waiting for the CXPACKET wait type.
4. The CXPACKET wait type indicates that parallel tasks are waiting for other tasks that are part of the same request to finish working.

▶ Task 4: Stop the Workload

- In the query pane, highlight the code under the heading for Task 4, and then click **Execute**. This will stop the workload.

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

Monitor workload pressure on schedulers.

Monitor thread status for user requests.

Exercise 3: Monitoring Waiting Tasks and Recording Wait Statistics

▶ Task 1: Clear Wait Statistics

1. In Solution Explorer, double-click the **Lab Exercise 03 - Waits.sql** query.
2. When the query window opens, in the query pane, type the following query after the Task 1 description:

```
DBCC SQLPERF('sys.dm_os_wait_stats', CLEAR);
```

3. Highlight the query, and then click **Execute**.

▶ Task 2: Check Current Wait Statistics

1. In the query pane, type the following under the heading for Task 2:

```
SELECT * FROM sys.dm_os_wait_stats;
```

2. Highlight the code that you have typed, and then click **Execute**.
3. Notice that most of the column values contain zero.

▶ Task 3: Start the Workload

1. Open Windows Explorer, and then navigate to D:\Labfiles\Lab01\Starter.
2. Right-click **start_load_exercise_03.ps1**, and then click **Run with PowerShell**. If prompted press **y** and then **Enter**. Leave the script running, and continue with the lab.

▶ Task 4: Monitor Waiting Tasks While the Workload Is Running

1. In SQL Server Management Studio, in the query pane, type the following under the heading for Task 4:

```
SELECT * FROM sys.dm_os_waiting_tasks WHERE session_id > 50;
```

2. Highlight the code that you have typed, and then click **Execute**.
3. You will see LCK_M_S waits in the **wait_type** column.

▶ Task 5: Record Wait Statistics for Analysis

1. Under the heading for Task 5, highlight the first query, and then click **Execute**.
2. Edit the second query under Task 5 so that it reads as follows:

```
SELECT ws.*
FROM #wait_stats_snapshot AS snap
JOIN sys.dm_os_wait_stats AS ws
ON ws.wait_type = snap.wait_type
WHERE ws.wait_time_ms - snap.wait_time_ms > 0
ORDER BY ws.wait_time_ms - snap.wait_time_ms DESC;
```

3. Highlight the code that you have amended, and then click **Execute**.

▶ Task 6: Stop the Workload

- In the query pane, highlight the code under the heading for Task 6, and then click **Execute**. This will stop the workload.

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

Monitor the waiting tasks list.

Capture and review wait statistics.

Lab 2: Testing Storage Performance

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler, and retailer of cycle products. The owners of the company have decided to start a new direct marketing arm of the company. It has been created as a new company named Proseware Inc. Even though Proseware Inc. has been set up as a separate company, it will receive some IT-related services from Adventure Works and will be provided with a subset of the corporate Adventure Works data. The existing Adventure Works SQL Server platform has been moved to a new server that can support both the existing workload and the workload from the new company.

Objectives

At the end of this lab, you will be able to configure and run Diskspd to test I/O subsystem performance.

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Configuring and Executing Diskspd

Scenario

You have reviewed wait statistics for the AdventureWorks database and noticed high wait statistics for I/O, among others. You want to make sure that I/O has been set up correctly and is performing optimally. In this exercise, you will use the Diskspd utility to test storage performance.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Execute Diskspd

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running.

2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

▶ Task 2: Execute Diskspd

1. On 10987C-MIA-SQL, you will find a copy of Diskspd.exe in D:\Labfiles\Lab02\Diskspd-v2.0.15\amd64fre.

2. Using this copy of the tool, run a test with the following parameters:

   - Duration: 3 minutes
   - Test file name: D:\Labfiles\Lab02\test.dat
   - Test file size: 2 GB
   - Thread count: 4
   - Percentage of writes: 40%
   - Block size: 64 KB
   - Outstanding I/O requests: 32
   - Read/write method: Random
   - Include Latency Statistics: Yes

Note: You will need to execute the tool as an administrator.

3. Review the output of the test.
4. Delete the test file (**D:\Labfiles\Lab02\test.dat**) when the test is complete.
5. Close Windows PowerShell®.

Results: At the end of this exercise, you will have configured and run Diskspd to test I/O subsystem performance.

Lab Answer Key 2: Testing Storage Performance

Exercise 1: Configuring and Executing Diskspd

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

▶ Task 2: Execute Diskspd

1. On 10987C-MIA-SQL, right-click the Start button, and then click **Windows PowerShell (Admin)**.
2. In the **User Account Control** dialog box, click **Yes**.
3. In the **Administrator: Windows PowerShell** window, type the following code, and then press Enter:

```
Cd D:\Labfiles\Lab02\Diskspd-v2.0.15\amd64fre
```

4. Type the following code, and then press Enter:

```
.\diskspd.exe -d180 -c2G -r -t4 -w40 -o32 -b64K -L D:\Labfiles\Lab02\test.dat;
```

5. After a few minutes, review the output of the test. Notice that the output includes:
   - CPU activity during the test, for each CPU.
   - Total I/O, read I/O, and write I/O statistics for each thread.
   - Total speed, read speed, and write speed by percentile.
6. When you have finished your review, delete the test file, by typing the following code, and then press Enter:

```
del D:\Labfiles\Lab02\test.dat
```

7. Close Windows PowerShell® when you have finished.

Results: At the end of this exercise, you will have configured and run Diskspd to test I/O subsystem performance.

Lab 3: Database Structures

Scenario

You have reviewed the AdventureWorks database and noticed high wait statistics for CPU, memory, I/O, blocking, and latching. In this lab, you will explore database structures and internals for a user database. You will enable instant file initialization and note the performance improvement. Finally, you will reduce **tempdb** latch contention by adding more data files to **tempdb**.

Objectives

After completing this lab, you will be able to:

- Explore database structures and data file internals.
- Improve performance by enabling instant file initialization.
- Reduce **tempdb** latch contention.

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Exploring Page Allocation Structure

Scenario

You have reviewed the AdventureWorks database and, amongst other things, noticed high wait statistics for I/O. Before investigating database data files and **tempdb** data files, you want to explore page and allocation structure.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Explore Page Structure
3. Explore Record Structure

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running, and then log on to **10987C-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: Explore Page Structure

1. Start Microsoft SQL Server Management Studio if it is not already running and connect to the **MIA-SQL** database instance using Windows Authentication.
2. Analyze page structure for the **Person.ContactType** table in the **AdventureWorks** database.

▶ Task 3: Explore Record Structure

- Turn on trace flag 3604 and use DBCC PAGE to analyze record structure for the **Person.ContactType** tablein the **AdventureWorks** database. See the Lab Answers sections if you want help with the SQL commands and queries.

Results: After completing this exercise, you will have explored data page and record structure.

Exercise 2: Configuring Instant File Initialization

Scenario

You have reviewed the AdventureWorks database and, amongst other things, noticed high wait statistics for I/O. One potential cause that you have identified is that instant file initialization is not being used. In this exercise, you will enable instant file initialization and record performance improvement.

The main tasks for this exercise are as follows:

1. Reset Security Policy

2. Record Workload Execution Time

3. Enable Instant File Initialization and Compare Run Time

▶ Task 1: Reset Security Policy

1. Use the **Local Security Policy** tool to identify which users have the **Perform volume maintenance** right.

2. Remove this right from the **Administrators** group.

▶ Task 2: Record Workload Execution Time

1. In SQL Server Management Studio, restart the SQL Server Services.

2. Open the file **InstantFileInit.sql** in the folder D:\Labfiles\Lab03\Starter and execute the code.

3. Note how long the script takes to complete.

▶ Task 3: Enable Instant File Initialization and Compare Run Time

1. Use the **Local Security Policy** tool to identify which users have the **Perform volume maintenance** right.

2. Add the right for the **Administrators** group.

3. Restart the SQL Server services, open the file **InstantFileInit.sql** in the folder D:\LabFiles\Lab03\starter if it is not already open, and then execute the code.

4. Note how long the script takes to complete.

5. Compare the run time of the script with and without instant file initialization enabled.

Results: At the end of this exercise, you will have enabled instant file initialization.

Exercise 3: Reconfiguring tempdb Data Files

Scenario

You have reviewed the AdventureWorks database and noticed, among other things, high wait statistics for latches. You have identified latch contention in **tempdb**. In this exercise, you will add more data files to **tempdb** to reduce latch contention.

The main tasks for this exercise are as follows:

1. Execute Workload and Record Latch Contention Metrics

2. Add Additional Data Files to tempdb

3. Measure Performance Improvement

▶ Task 1: Execute Workload and Record Latch Contention Metrics

1. Execute the script **tempdbLoad.cmd D:\Labfiles\Lab03\Starter** as an administrator.
2. When all the command windows have closed, use the **sys.dm_os_wait_stats dmv** to record the LATCH waits for the **MIA-SQL** server instance.

▶ Task 2: Add Additional Data Files to tempdb

- Open the file **addTempdbFiles.sql** in SQL Server Management Studio and execute the code to create seven additional **tempdb** data files.

▶ Task 3: Measure Performance Improvement

1. Compare the wait stats figures before and after the additiona**l tempdb** files were added, noting the reduced waits with more **tempdb** files.
2. Close SQL Server Management Studio without saving changes.

Results: After completing this lab, **tempdb** will be using multiple data files.

Lab Answer Key 3: Database Structures

Exercise 1: Exploring Page Allocation Structure

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running, and then log on to **10987C-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 wait for the script to finish.

▶ Task 2: Explore Page Structure

1. On the taskbar, click **Microsoft SQL Server Management Studio 17**.
2. In the **Connect to Server** dialog box, click **Connect**.
3. Click **New Query**, type the following Transact-SQL, and then click **Execute**:

```
USE AdventureWorks;
GO
SELECT db_name(database_id) Database_Name, object_name([object_id]) Table_Name,
allocation_unit_type, allocation_unit_type_desc
  allocated_page_file_id, allocated_page_page_id, page_type, page_type_desc FROM
sys.dm_db_database_page_allocations(db_id('AdventureWorks'),object_id('Person.Contact
Type'),NULL,NUll,'DETAILED');
GO
```

4. Examine the query results, noting that there are four pages: two IAM pages, an index page, and a data page. Note the value in the **allocated_page_page_id** column for the row with the value **DATA_PAGE** in the **page_type_desc** column.

▶ Task 3: Explore Record Structure

1. In the query window, type the following Transact-SQL to enable trace flag 3604, highlight the code, and then click **Execute**:

```
DBCC TRACEON(3604);
GO
```

2. Type the following Transact-SQL, replacing the characters **XXX** with the page id you noted in the previous lab task; highlight the code, and then click **Execute**:

```
DBCC PAGE(11,1,XXX,2)
GO
```

3. Examine the query results, noting the page type, allocation status and other page information.

Results: After completing this exercise, you will have explored data page and record structure.

Exercise 2: Configuring Instant File Initialization

▶ Task 1: Reset Security Policy

1. Click **Start**, type **secpol.msc**, and then click **secpol.msc**.
2. In the **Local Security Policy** management console, in the left pane, under **Security Settings**, expand **Local Policies**, and then click **User Rights Assignment**.
3. In the right pane, under **Policy**, double-click **Perform volume maintenance tasks**.
4. In the **Perform volume maintenance tasks Properties** dialog box, on the **Local Security Setting** tab, click **Administrators**, click **Remove**, and then click **OK**.
5. Leave the **Local Security Policy** management console open for use later in the lab.

▶ Task 2: Record Workload Execution Time

1. In SQL Server Management Studio, in Object Explorer, right-click the **MIA-SQL** database instance, and click **Restart**.
2. In the **User Account Control** dialog box, click **Yes**.
3. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes**.
4. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes** to confirm you want to restart dependent services.
5. When the services have restarted, on the **File** menu, point to **Open**, and then click **File**.
6. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab03\Starter** folder, and then double-click **InstantFileInit.sql**.
7. Click **Execute** and note how long the script takes to run.
8. Leave SQL Server Management Studio open.

▶ Task 3: Enable Instant File Initialization and Compare Run Time

1. In the Local Security Policy management console, in the right pane, under **Policy**, double-click **Perform volume maintenance tasks**.
2. In the **Perform volume maintenance tasks Properties** dialog box, on the **Local Security Setting** tab, click **Add User or Group**.
3. In the **Select Users, Computers, Service Accounts, or Groups** dialog box, in the **Enter the object names to select** box, type **Administrators**, and then click **OK**.
4. In the **Perform volume maintenance tasks Properties** dialog box, click **OK**.
5. In SQL Server Management Studio, in Object Explorer, right-click the **MIA-SQL** database instance, and then click **Restart**.
6. In the **User Account Control** dialog box click **Yes**.
7. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes**.
8. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes** to confirm you want to restart dependent services.
9. When SQL Server services have restarted, click **Execute** to run the code in the query window again.
10. Compare the run time of the script with and without instant file initialization enabled.
11. Close SQL Server Management Studio, without saving any changes.

Results: At the end of this exercise, you will have enabled instant file initialization.

Exercise 3: Reconfiguring tempdb Data Files

▶ Task 1: Execute Workload and Record Latch Contention Metrics

1. In the **D:\Labfiles\Lab03\Starter** folder, right-click the **tempdbLoad.cmd** file, and then click **Run as administrator**.
2. In the **User Account Control** dialog box, click **Yes**.
3. Wait until all the command windows have closed.
4. On the taskbar, click **Microsoft SQL Server Management Studio 17**.
5. In the **Connect to Server** dialog box, click **Connect**.
6. Click **New Query**, type the following Transact-SQL, and then click **Execute**:

```
SELECT *
FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'PAGELATCH%'
```

7. Note the wait stats for the SQL Server instance.

▶ Task 2: Add Additional Data Files to tempdb

1. On the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, in the **D:\Labfiles\Lab03\Starter** folder, double-click **addTempdbFiles.sql**.
3. Click **Execute** to run the code.

▶ Task 3: Measure Performance Improvement

1. In the **D:\Labfiles\Lab03\Starter** folder, right-click the **tempdbLoad.cmd** file, and then click **Run as administrator**.
2. In the **User Account Control** dialog box, click **Yes**.
3. Wait until all the command windows have closed.
4. In SQL Server Management Studio, click **New Query**, type the following Transact-SQL, and then click **Execute**:

```
select *
from sys.dm_os_wait_stats
where wait_type like 'PAGELATCH%'
```

5. Note the wait stats for the SQL server instance.
6. Compare the previous wait stats figures with those from before the additional **tempdb** files were added, noting the reduced waits with more **tempdb** files.
7. Close SQL Server Management Studio without saving changes.

Results: After completing this lab, **tempdb** will be using multiple data files.

Lab 5: Concurrency and Transactions

Scenario

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

Objectives

After completing this lab, you will be able to:

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

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Implement Snapshot Isolation

Scenario

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

The main tasks for this exercise are as follows:

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

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab05\Starter** folder as Administrator.

▶ Task 2: Clear Wait Statistics

1. Start **SQL Server Management Studio** and connect to the MIA-SQL instance using Windows authentication; then open the project file **D:\Labfiles\Lab05\Starter\Project\Project.ssmssln** and the script file **Lab Exercise 01 - snapshot isolation.sql**.
2. Execute the query under the comment that begins **Task 1** to clear wait statistics.

▶ Task 3: Run the Workload

- In the **D:\Labfiles\Lab05\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell™. Wait for the workload to finish before continuing. If a message is displayed asking you to confirm a change in execution policy, type **Y**.

▶ Task 4: Capture Lock Wait Statistics

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

▶ Task 5: Enable SNAPSHOT Isolation

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

▶ Task 6: Implement Snapshot Isolation

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

▶ Task 7: Rerun the Workload

1. In the SSMS query window for **Lab Exercise 01 - snapshot isolation.sql,** rerun the query under the comment that begins **Task 1**.
2. In the **D:\Labfiles\Lab05\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell. Wait for the workload to finish before continuing.

▶ Task 8: Capture New Lock Wait Statistics

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

▶ Task 9: Compare Overall Lock Wait Time

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

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

Exercise 2: Implement Partition Level Locking

Scenario

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

The main tasks for this exercise are as follows:

1. Open Activity Monitor

2. Clear Wait Statistics

3. View Lock Waits in Activity Monitor

4. Enable Partition Level Locking

5. Rerun the Workload

▶ Task 1: Open Activity Monitor

1. In SSMS Object Explorer, open Activity Monitor for the **MIA-SQL** instance.

2. In Activity Monitor, expand the **Resource Waits** section.

▶ Task 2: Clear Wait Statistics

1. If it is not already open, open the project file **D:\Labfiles\Lab05\Starter\Project\Project.ssmssln**, then open the query file **Lab Exercise 02 - partition isolation.sql**.

2. Execute the code under **Task 2** to clear wait statistics.

▶ Task 3: View Lock Waits in Activity Monitor

1. In the **D:\Labfiles\Lab05\Starter** folder, execute **start_load_exercise_02.ps1** with PowerShell. Wait for the workload to finish before continuing (it will take a few minutes to complete).

2. Switch to SSMS and to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.

3. Close the PowerShell window where the workload was executed.

▶ Task 4: Enable Partition Level Locking

1. Return to the query window where **Lab Exercise 02 - partition isolation.sql** is open.

2. Under the comment that begins **Task 5**, write a query to alter the **Proseware.CampaignResponsePartitioned** table in the **AdventureWorks** database to enable partition level locking.

3. Rerun the query under the comment that begins **Task 2** to clear wait statistics.

▶ Task 5: Rerun the Workload

1. In the **D:\Labfiles\Lab05\Starter** folder, execute **start_load_exercise_02.ps1** with PowerShell. Wait for the workload to finish before continuing (it will take a few minutes to complete).

2. Return to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec**) for the **Lock** wait type.

3. Compare this value to the value you noted earlier in the exercise.

4. Close the PowerShell window where the workload was executed.

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

Lab Answer Key 5: Concurrency and Transactions

Exercise 1: Implement Snapshot Isolation

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. In the **D:\Labfiles\Lab05\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Clear Wait Statistics

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab05\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click the query **Lab Exercise 01 - snapshot isolation.sql**. (If Solution Explorer is not visible, select **Solution Explorer** on the **View** menu or press Ctrl+Alt+L on the keyboard.)
5. To clear wait statistics, select the query under the comment that begins **Task 1**, and then click **Execute**.

▶ Task 3: Run the Workload

1. Open Windows Explorer and navigate to the **D:\Labfiles\Lab05\Starter** folder.
2. Right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**.
3. If a message is displayed asking you to confirm a change in execution policy, type **Y** and then press ENTER.
4. Wait for the workload to complete and then press ENTER to close the window.

▶ Task 4: Capture Lock Wait Statistics

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

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

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

▶ Task 5: Enable SNAPSHOT Isolation

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

▶ Task 6: Implement Snapshot Isolation

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

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

3. Click **Execute**.

▶ Task 7: Rerun the Workload

1. In the SQL Server Management Studio query window for **Lab Exercise 01 - snapshot isolation.sql**, select the query under the comment that begins **Task 1**, and then click **Execute**.
2. Switch to File Explorer and in the **D:\Labfiles\Lab05\Starter** folder, right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**.
3. Wait for the workload to complete.

▶ Task 8: Capture New Lock Wait Statistics

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

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

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

▶ Task 9: Compare Overall Lock Wait Time

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

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

Exercise 2: Implement Partition Level Locking

▶ Task 1: Open Activity Monitor

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

▶ Task 2: Clear Wait Statistics

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

▶ Task 3: View Lock Waits in Activity Monitor

1. Switch to File Explorer and in the **D:\Labfiles\Lab05\Starter** folder, right-click **start_load_exercise_02.ps1**, and then click **Run with PowerShell**.
2. Wait for the workload to complete (it will take a few minutes).
3. Switch to SQL Server Management Studio and to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.
4. Switch to the PowerShell workload window and press Enter to close it.

▶ Task 4: Enable Partition Level Locking

1. Return to the SQL Server Management Studio query window where **Lab Exercise 02 - partition isolation.sql** is open.
2. Under the comment that begins **Task 5**, type:

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

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

▶ Task 5: Rerun the Workload

1. Switch to File Explorer and in the **D:\Labfiles\Lab05\Starter** folder, right-click **start_load_exercise_02.ps1** and then click **Run with PowerShell**.
2. Wait for the workload to complete (it will take a few minutes).
3. Return to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.
4. Compare this value to the value you noted earlier in the exercise; the wait time will be considerably lower after you implemented partition level locking.
5. Switch to the PowerShell workload window and press Enter to close it.

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

Lab 6: Statistics and Index Internals

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler and retailer of cycle products. The owners have decided to start a new direct marketing arm of the company. It has been created as a new company named Proseware Inc. Even though it has been set up as a separate company, it will receive some IT-related services from Adventure Works and will be provided with a subset of the corporate Adventure Works data. The existing Adventure Works SQL Server platform has been moved to a new server that is capable of supporting both the existing workload and the workload from the new company.

While investigating the general slow speed of the new SQL Server instance, you came across a few workloads with poor execution performance, due to cardinality estimation issues and inappropriate indexes. In this lab, you will improve the performance of those workloads by fixing cardinality estimation errors and creating the correct indexes, including columnstore indexes.

Objectives

After completing this lab, you will be able to:

- Identify and fix cardinality estimation errors.
- Identify and review the indexing strategy.
- Create columnstore indexes.

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Fixing Cardinality Estimation Errors

Scenario

While investigating the general slow speed of the new SQL Server instance, you came across a few workloads that had cardinality estimation issues. In this exercise, you will fix cardinality estimation error for this type of workload and improve the performance.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Run the Workload

3. List Statistics Objects

4. Examine Statistics in Detail

5. Update Statistics

6. Rerun the Workload

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

2. Run **Setup.cmd** as Administrator in the **D:\Labfiles\Lab06\Starter** folder.

▶ Task 2: Run the Workload

1. In the **D:\Labfiles\Lab06\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell™. If a message is displayed asking you to confirm a change in execution policy, type **Y**. Wait a few minutes for the workload to finish.
2. Note the elapsed time reported by the script.

▶ Task 3: List Statistics Objects

1. Start **SQL Server Management Studio (SSMS)** and connect to the **MIA-SQL** instance using Windows authentication.
2. Open the project file **D:\Labfiles\Lab06\Starter\Project\Project.ssmssln** and the script file **Lab Exercise 01 - Cardinality.sql**.
3. Execute the query under the comment that begins **Task 2** to list the statistics objects for the **Proseware.WebResponse**, **Proseware.Campaign**, and **Proseware.CampaignAdvert** tables in the **AdventureWorks** database.
4. Do any of the statistics look like they might be out of date?

▶ Task 4: Examine Statistics in Detail

1. Under the comment that begins **Task 3**, amend and then execute the first query to display the detailed statistics information for the **IX_WebResponse_CampaignAdvertID** statistics object linked to **Proseware.WebResponse**. How many rows does the table contain, according to the statistics?
2. Execute the second query to find the actual number of rows in the **Proseware.WebResponse** table. Do these results suggest that the query might be subject to a cardinality estimation error?

▶ Task 5: Update Statistics

1. Under the comment that begins **Task 4**, amend and then execute the first query to update all the statistics objects linked to **Proseware.Webresponse** by sampling all of the rows in the table.
2. Execute the second query under the comment that begins **Task 4** to examine the new statistics for the **IX_WebResponse_CampaignAdvertID** statistics object.
3. Amend and then execute the third query under the comment that begins **Task 4** to update all the statistics objects linked to **Proseware.CampaignAdvert** using 50 percent of the rows in the table.

▶ Task 6: Rerun the Workload

1. In the **D:\Labfiles\Lab06\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell. Wait for the workload to finish before continuing.
2. Compare the elapsed time reported by the script to the elapsed time reported in the first step of this exercise.

Results: At the end of this lab, statistics in the **AdventureWorks** database will be updated.

Exercise 2: Improve Indexing

Scenario

While investigating the general slow speed of the new SQL Server instance, you came across a few workloads that did not have the correct indexes in the underlying tables. In this exercise, you will modify existing indexes to improve the performance of the workload.

The main tasks for this exercise are as follows:

1. Execute the Workload
2. Examine Existing Indexes
3. Use the Database Engine Tuning Advisor
4. Implement a Covering Index
5. Rerun the Workload

▶ Task 1: Execute the Workload

- Execute the Transact-SQL script in the **Lab Exercise 02 - Workload.sql** file.

▶ Task 2: Examine Existing Indexes

1. In SSMS Solution Explorer, open the script file **Lab Exercise 02 - Indexing.sql**.
2. Under the comment that begins **Task 2**, write a query to return details of the indexes on the **Proseware.WebResponse** table in the **AdventureWorks** database.

▶ Task 3: Use the Database Engine Tuning Advisor

1. In SSMS, start the Database Engine Tuning Advisor from the **Tools** menu, and connect to the **MIA-SQL** instance using Windows authentication.
2. Run a performance analysis session against the **AdventureWorks** database using the file **D:\Labfiles\Lab06\Starter\Project\Project\ Lab Exercise 02 - Workload.sql** as a workload file.
3. When the analysis completes, examine the definition of the index suggested for the **Proseware.WebResponse** table, but do not implement it. Close the Database Engine Tuning Advisor when you are finished. Is the suggested index similar to any of the existing indexes on the table?

▶ Task 4: Implement a Covering Index

1. In SSMS, in the **Lab Exercise 02 - Indexing.sql** file, under the comment that begins **Task 5**, execute the first query to drop the index **IX_WebResponse_log_date_CampaignAdvertID**.
2. Amend and then execute the second query under the comment that begins **Task 5** to create a new index with the key columns:
   - **log_date**
   - **CampaignAdvertID**
   - **browser_name**
3. and include: **page_visit_time_seconds**

▶ Task 5: Rerun the Workload

1. In SSMS, execute the script in the **Lab Exercise 02 - Workload.sql** file.
2. Return to the **Lab Exercise 02 - Indexing.sql** file, and under the comment that begins **Task 6**, execute the query against the system DMV **sys. dm_db_index_usage_stats** to verify that the new index was used.

Results: At the end of this exercise, the indexing of the **Proseware.WebResponse** table in the **AdventureWorks** database will be improved.

Exercise 3: Using Columnstore Indexes

Scenario

The **Proseware.WebResponse** table is expected to grow significantly in the future. Because the table will be used in many range-based analytical queries, you decide to create a nonclustered columnstore index on some of the columns in the table. You will also create a new table, **Proseware.Demographic**, to hold a very large dataset; this table will have a clustered columnstore index.

The main tasks for this exercise are as follows:

1. Add a Nonclustered Columnstore Index to a Row-Based Table
2. Create a Table with a Clustered Columnstore Index
3. Add a Nonclustered Row-Based Index to a Table with a Clustered Columnstore Index

▶ Task 1: Add a Nonclustered Columnstore Index to a Row-Based Table

1. In SSMS Solution Explorer, open the script file **Lab Exercise 03 - Columnstore.sql**.
2. Under the comment that begins **Task 1**, amend and then execute the query to add a nonclustered columnstore index to **Proseware.WebResponse** in the **AdventureWorks** database. The columnstore index should cover the following columns:
   - log_date
   - page_url
   - browser_name
   - page_visit_time_seconds
3. Name the index **IX_NCI_WebResponse**.

▶ Task 2: Create a Table with a Clustered Columnstore Index

1. Under the comment that begins **Task 2,** amend and then execute the query to create a table called **Proseware.Demographic** with a clustered columnstore index called **PK_Proseware_Weblog**.
2. Execute the second statement under the comment that begins **Task 2** to add a single row to **Proseware.Demographic**.
3. Query the table to verify that one row has been inserted.

▶ Task 3: Add a Nonclustered Row-Based Index to a Table with a Clustered Columnstore Index

1. Under the comment that begins **Task 3**, amend and then execute the first query to add a nonclustered unique index to **Proseware.Demographic**. Call the index **IX_Demographic_DemographicID**. The index key should be **DemographicID**.

2. Execute the second statement under the comment that begins **Task 3** to rerun the INSERT statement from the previous step. Notice that the nonclustered index prevents you from inserting duplicate data.

Results: At the end of this exercise, the **Proseware.WebResponse** in the **AdventureWorks** database will have a nonclustered columnstore index. A new table—**Proseware.Demographic**—will be created with a clustered columnstore index.

Lab Answer Key 6: Statistics and Index Internals

Exercise 1: Fixing Cardinality Estimation Errors

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Using Windows Explorer, navigate to the **D:\Labfiles\Lab06\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: Run the Workload

1. Open File Explorer and navigate to **D:\Labfiles\Lab06\Starter**.
2. Right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**. If a message is displayed asking you to confirm a change in execution policy, type **Y** and then press ENTER.
3. Wait a few minutes for the workload to complete.
4. Note the elapsed time reported by the script, and then press ENTER to close the PowerShell window.

▶ Task 3: List Statistics Objects

1. Start **SQL Server Management Studio (SSMS)** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab06\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click **Lab Exercise 01 - Cardinality.sql**. (If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**.)
5. Select the query under the comment that begins **task 2** and click **Execute**.
6. The following statistics objects look like they might be out of date:
   - **Proseware.CampaignAdvert.IX_CampaignAdvert_AdvertMedia**
   - **Proseware.WebResponse.IX_WebResponse_CampaignAdvertID**

▶ Task 4: Examine Statistics in Detail

1. Edit the first query under the comment that begins **Task 3** so that it reads:

```
DBCC SHOW_STATISTICS
('Proseware.WebResponse','IX_WebResponse_CampaignAdvertID');
```

2. Select the query you have amended and click **Execute**. Note the value of the **rows** column in the first result set.
3. Select the second query under the comment that begins **Task 3** and click **Execute**. Compare the value returned by the query to the value you noted in step 2.
4. The values are substantially different: 1,000 compared to 1 million. This degree of error in the statistics is likely to cause a cardinality estimation problem.

▶ Task 5: Update Statistics

1. Edit the first query under the comment that begins **Task 4**, so that it reads:

```
UPDATE STATISTICS Proseware.WebResponse WITH FULLSCAN;
```

2. Select the query that you have amended and click **Execute**.
3. Select the second query under the comment that begins **Task 4** and click **Execute**. The query has the following text:

```
DBCC SHOW_STATISTICS ('Proseware.WebResponse','IX_WebResponse_CampaignAdvertID');
```

4. Edit the third query under the comment that begins **Task 4** so that it reads:

```
UPDATE STATISTICS Proseware.CampaignAdvert WITH SAMPLE 50 PERCENT;
```

5. Select the query that you have amended and click **Execute**.

▶ Task 6: Rerun the Workload

1. In File Explorer, navigate to **D:\Labfiles\Lab06\Starter**.
2. Right-click **start_load_exercise_01.ps1** and then click **Run with PowerShell**.
3. Wait for the workload to complete.
4. Note the elapsed time reported by the script, and then press ENTER to close the PowerShell window.
5. With updated statistics, the execution of the workload is considerably faster.

Results: At the end of this lab, statistics in the **AdventureWorks** database will be updated.

Exercise 2: Improve Indexing

▶ Task 1: Execute the Workload

1. In SQL Server Management Studio, in Solution Explorer, double-click **Lab Exercise 02 - Workload.sql**.
2. Click **Execute**.

▶ Task 2: Examine Existing Indexes

1. In Solution Explorer, double-click **Lab Exercise 02 - Indexing.sql**.
2. Under the comment that begins **Task 2**, type:

```
USE AdventureWorks;
EXEC sp_help 'Proseware.WebResponse';
```

3. Highlight the code you have written, and then click **Execute**.
4. Information about indexes on the table will be found in the sixth result set in the Results pane.

▶ Task 3: Use the Database Engine Tuning Advisor

1. In SQL Server Management Studio, on the **Tools** menu, click **Database Engine Tuning Advisor**.
2. When the tuning advisor starts, in the **Connect to Server** dialog box, connect to the **MIA-SQL** database engine using Windows authentication.
3. In the **Workload** section, click **File**, in the text box, type **D:\Labfiles\Lab06\Starter\Project\Project\Lab Exercise 02 - Workload.sql**, and then in the **Database for workload analysis** box, click **AdventureWorks**.
4. In the **Select databases and tables to tune** list, select **AdventureWorks**. On the row for the **AdventureWorks** database, click in the **Selected Tables** column, click the drop-down arrow, clear the check box next to **Name**, and then select **WebResponse**. Click the drop-down arrow again to close the list.
5. On the toolbar, click **Start Analysis**.
6. When analysis completes, on the **Index Recommendations** tab, click the value in the **Definition** column starting **([log_date] asc** to examine the suggested index definition. Notice that the suggested index is similar to the **IX_WebResponse_log_date_CampaignAdvertID** index already on the table.
7. Close the Database Engine Tuning Advisor.

▶ Task 4: Implement a Covering Index

1. In SQL Server Management Studio, switch to the query pane where the **Lab Exercise 02 - Indexing.sql** file is open.
2. Highlight the first query under the comment that begins **Task 5**, then click **Execute** to drop the existing index.
3. Edit the second query under the comment that begins **Task 5** to read:

```
CREATE INDEX IX_WebResponse_log_date_CampaignAdvertID_browser_name
ON Proseware.WebResponse (log_date, CampaignAdvertID,browser_name)
INCLUDE (page_visit_time_seconds);
```

4. Highlight the query you have amended, and then click **Execute**.

▶ Task 5: Rerun the Workload

1. In SQL Server Management Studio, switch to **Lab Exercise 02 - Workload.sql**.
2. Click **Execute**.
3. Switch to **Lab Exercise 02 - Indexing.sql**. Select the query under the comment that begins **Task 6** and click **Execute**. The results of the query demonstrate that the new index was used.
4. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, the indexing of the **Proseware.WebResponse** table in the **AdventureWorks** database will be improved.

Exercise 3: Using Columnstore Indexes

▶ Task 1: Add a Nonclustered Columnstore Index to a Row-Based Table

1. In SQL Server Management Studio, in Solution Explorer, double-click **Lab Exercise 03 - Columnstore.sql**.
2. Amend the query under the comment that begins **Task 1** so that it reads:

```
USE AdventureWorks;
GO
CREATE COLUMNSTORE INDEX IX_NCI_WebResponse
ON Proseware.WebResponse (log_date, page_url, browser_name, page_visit_time_seconds);
GO
```

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

▶ Task 2: Create a Table with a Clustered Columnstore Index

1. Amend the first query under the comment that begins **Task 2** so that it reads:

```
CREATE TABLE Proseware.Demographic
(  DemographicID bigint NOT NULL,
             DemoCode varchar(50),
             Code01 int,
             Code02 int,
             Code03 tinyint,
             Income decimal(18,3),
             MaritalStatus char(1),
             Gender char(1),
             INDEX PK_Proseware_Weblog CLUSTERED COLUMNSTORE
);
```

2. Select the query that you have edited and click **Execute**.
3. Select the second query under the comment that begins **Task 2** and click **Execute**.
4. After the query that you executed in the previous step, type:

```
SELECT * FROM Proseware.Demographic;
```

5. Highlight the query you have typed and click **Execute**. Verify that one row has been inserted.

▶ Task 3: Add a Nonclustered Row-Based Index to a Table with a Clustered Columnstore Index

1. Edit the first query under the comment that begins **Task 3** so that it reads:

```
CREATE UNIQUE NONCLUSTERED INDEX IX_Demographic_DemographicID
ON Proseware.Demographic (DemographicID);
```

2. Select the query you have amended and click **Execute**.
3. Select the second query under the comment that begins **Task 3** and click **Execute**.

   Note that an error is raised; this is expected behavior, because the nonclustered index prevents you from inserting duplicate data.

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

Results: At the end of this exercise, the **Proseware.WebResponse** in the **AdventureWorks** database will have a nonclustered columnstore index. A new table—**Proseware.Demographic**—will be created with a clustered columnstore index.

Lab 7: Query Execution and Query Plan Analysis

Scenario

While investigating a new SQL Server instance, you have come across some workloads that are running slowly. You decide to analyze execution plans for those workloads. In this lab, you will analyze execution plans and identify plan issues. You will then fix them to improve execution performance of workloads.

Objectives

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

- Improve the performance of a SELECT statement by analyzing the query plan.
- Improve the performance of a stored procedure by analyzing the query plan.

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Improving SELECT Performance for Historical Marketing Campaign Data

Scenario

The Proseware Inc. team has acquired some historical data about marketing campaigns run by a similar company. This data has been loaded into the Proseware schema in the AdventureWorks database.

Data users are complaining that their queries are running slowly, and have provided an example of a query which runs more slowly than they expect. Your task is to examine the query plan for clues to the source of its poor performance.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Collect an Actual Execution Plan

3. Rebuild Table Statistics

4. Compare the New Actual Execution Plan

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQ**L as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab07\Starter** folder as Administrator.

▶ Task 2: Collect an Actual Execution Plan

1. Start **SQL Server Management Studio**, then open the project file **D:\Labfiles\Lab07\Starter\Project\Project.ssmssln** and the Transact-SQL file **Lab Exercise 01 - tuning 1.sql**.
2. Collect an actual execution plan for the query shown under the comment that begins **Task 1**. Note the execution time.

3. Save the actual query plan as **D:\Labfiles\Lab07\plan1.sqlplan**.
   Looking at the actual query execution plan, do you see any potential causes of the performance issue? Hint: compare estimated row count to actual row count for several of the operators in the query plan.

▶ Task 3: Rebuild Table Statistics

- Execute the query under the comment that begins **Task 2** to rebuild the statistics held for the **Proseware.Campaign** and **Proseware.CampaignResponse** tables.

▶ Task 4: Compare the New Actual Execution Plan

1. Re-run the query under the comment that begins **Task 1** and collect a new actual execution plan.
2. Compare the new execution plan to the plan you saved in **Task 1** (as **D:\Labfiles\Lab07\plan1.sqlplan**).
3. Is the run time of the query reduced? Does the actual row count now match the estimated row count more closely?
4. Are there more improvements you could make to the performance of this query based on the data returned in the execution plan?

Results: At the end of this exercise, you will have improved the performance of a SELECT query by analyzing the query plan.

Exercise 2: Improving Stored Procedure Performance

Scenario

The Proseware Inc. team has decided to continue to add new data to the **Proseware.CampaignResponse** table. A stored procedure—**Proseware.up_CampaignResponse_Add**—has been created for this purpose. Because the Proseware Inc. team expects to be calling this stored procedure many times a day, you will check the performance of the stored procedure code by examining the query plan. For the purposes of this exercise, you can add rows to the campaign named **1010000**, because this campaign name is being used for testing.

The main tasks for this exercise are as follows:

1. Collect an Actual Execution Plan

2. Add a Covering Index

3. Change the Data Type of the @CampaignName Parameter

▶ Task 1: Collect an Actual Execution Plan

1. In SSMS Solution Explorer, open the Transact-SQL file **Lab Exercise 02 - tuning 2.sql**.
2. Amend the query under the comment that begins **Task 1**, to execute the stored procedure **Proseware.up_CampaignResponse_Add** with the following parameter values:
   - @CampaignName = 1010000
   - @ResponseDate = '2016-03-01'
   - @ConvertedToSale = 1
   - @ConvertedSaleValueUSD = 100.00

3. Execute the query, collecting an actual execution plan.
4. Save the actual query plan as **D:\Labfiles\Lab07\plan2.sqlplan**.

   What two changes might you suggest, to the database or the stored procedure code, to improve the performance of this stored procedure?

▶ Task 2: Add a Covering Index

1. Under the heading for **Task 2**, edit the query to add a unique nonclustered index to the **Proseware.Campaign** table on the **CampaignName** column.
2. Execute the code under the **Task 1** heading again. Save the actual query plan as **D:\Labfiles\Lab07\plan3.sqlplan**.

   What do you notice about the new plan compared to the plan you collected in **Task 1** (saved as **D:\Labfiles\Lab07\plan2.sqlplan**)? Why is an index scan used on ix_Campaign_CampaignName in the first query?

▶ Task 3: Change the Data Type of the @CampaignName Parameter

1. Under the comment that begins **Task 3**, amend the query to change the data type of the @CampaignName parameter of the stored procedure **Proseware.up_CampaignResponse_Add** to **varchar(20)**.
2. Execute the code under the **Task 1** heading again. Save the actual query plan as **D:\Labfiles\Lab07\plan4.sqlplan**.
   What do you notice about the new plan compared to the plan you collected in **Task 1** (saved as **D:\Labfiles\Lab07\plan2.sqlplan**)?

Results: At the end of this lab, you will have examined a query execution plan for a stored procedure and implemented performance improvements by adding a covering index and eliminating an implicit data type conversion.

Lab Answer Key 7: Query Execution and Query PlanAnalysis

Exercise 1: Improving SELECT Performance for Historical Marketing Campaign Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQ**L as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab07\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: Collect an Actual Execution Plan

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows® authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab07\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click the **Lab Exercise 01 - tuning 1.sql**.
5. Select the query under the comment which begins **Task 1**. On the **Query** menu, click **Include Actual Execution Plan**.
6. Click **Execute**. Note the execution time.
7. In the Results pane, on the **Execution plan** tab, right-click the graphical execution plan, and then click **Save Execution Plan As**.
8. In the **Save As** dialog box, enter the file name as **D:\Labfiles\Lab07\plan1.sqlplan** and then click **Save**.
9. On the **Execution plan** tab, scroll to the far right-hand side of the actual query plan. In the top right-hand corner, position the cursor over the query plan operator, the name of which starts **Clustered Index Scan (Clustered)**. In the pop-up that appears, notice the difference between **Estimated Number of Rows** (approximately 3.74) and **Actual Number of Rows** (more than 1.8 million).

   This type of issue is caused by a poor estimate of cardinality. The table statistics indicate that the tables involved have very few rows, but in reality, the row count is much higher. Updating statistics for the tables involved will improve performance.

▶ Task 3: Rebuild Table Statistics

1. In SQL Server Management Studio, in the query pane, select the query under the comment that begins **Task 2**:

```
ALTER TABLE Proseware.Campaign REBUILD
GO
ALTER TABLE Proseware.CampaignResponse REBUILD;
GO
```

2. Click **Execute**.

▶ Task 4: Compare the New Actual Execution Plan

1. In the query pane, select the query under the comment that begins **Task 1** and click **Execute**. Note the run time of the query.
2. In the Results pane, on the **Execution plan** tab, scroll to the far right of the actual query plan. In the top right, position the cursor over the query plan operator, the name of which starts **Clustered Index Scan (Clustered)…**. In the pop-up that appears, notice the similarity in the **Estimated Number of Rows** (more than 1.8 million) and **Actual Number of Rows** (more than 1.8 million).
3. Note that the estimated and actual row counts now match almost exactly. The query will execute faster than it did previously.
4. The execution plan includes a suggestion for an index that will improve query performance.
5. On the **Execution plan** tab, right-click the graphical execution plan and click **Compare Showplan**. In the **Open** dialog box, select **D:\Labfiles\Lab07\plan1.sqlplan** and click **Open**. The new and old query execution plans will open side-by-side.
6. When you have finished your comparison, close the **Showplan Comparison** tab. Leave SSMS open for the next exercise.

Results: At the end of this exercise, you will have improved the performance of a SELECT query by analyzing the query plan.

Exercise 2: Improving Stored Procedure Performance

▶ Task 1: Collect an Actual Execution Plan

1. In SQL Server Management Studio, in Solution Explorer, double-click **Lab Exercise 02 - tuning 2.sql**.
2. In the query pane, highlight the following text, and then click **Execute**:

```
USE AdventureWorks;
GO
```

3. On the **Query** menu, click **Include Actual Execution Plan**.
4. In the query pane, under the comment that begins **Task 1**, edit the query so that it reads:

```
EXEC Proseware.up_CampaignResponse_Add
 @CampaignName = 1010000,
 @ResponseDate = '2016-03-01',
 @ConvertedToSale = 1,
 @ConvertedSaleValueUSD = 100.00;
```

5. Select the query you have edited then click **Execute**.
6. In the Results pane, on the **Execution plan** tab, right-click the graphical execution plan, and then click **Save Execution Plan As**.
7. In the **Save As** dialog box, use the file name **D:\Labfiles\Lab07\plan2.sqlplan**, then click **Save**.
8. On the **Execution plan** tab, notice that the first query in the batch has 77 percent of the total batch cost. Notice the execution plan warning on the SELECT operator in the first query, and that the selection of data from the **Proseware.Campaign** table uses an index scan.
9. There are two changes that might improve the performance of this query:
   a. You could add a nonclustered index to the **Proseware.Campaign.CampaignName** column.
   b. You could change the data type of the @CampaignName parameter of the stored procedure so that it matches the data type of the **Proseware.Campaign.CampaignName** column.

▶ Task 2: Add a Covering Index

- In the query pane, amend the query under the comment that begins **Task 2** so that it reads:

```
CREATE UNIQUE NONCLUSTERED INDEX ix_Campaign_CampaignName ON Proseware.Campaign
(CampaignName);
```

2. Select the query you have edited and click **Execute**.
3. Select the code under the comment that begins **Task 1** and click **Execute**.
4. In the Results pane, on the **Execution plan** tab, right-click the **graphical execution plan** and click **Save Execution Plan As**.
5. In the **Save As** dialog box, use the file name **D:\Labfiles\Lab07\plan3.sqlplan** and click **Save**.
6. Notice that the relative cost of the first query in the batch has reduced slightly, and that a scan of the new index is being used.
7. An index scan is used because of the data type mismatch between the @CampaignName parameter and the **CampaignName** column. The warning on the SELECT operator is still present.

▶ Task 3: Change the Data Type of the @CampaignName Parameter

1. In the query pane, under the comment that begins **Task 3**, edit the first three lines of the text in the stored procedure definition so that they read as follows:

```
ALTER PROCEDURE Proseware.up_CampaignResponse_Add
(
 @CampaignName varchar(20),
```

2. Select the query under the comment that begins **Task 3**—from ALTER PROCEDURE… to the end of the script—then click **Execute**.
3. Select the query under the comment that begins **Task 1** and click **Execute**.
4. In the Results pane, on the **Execution plan** tab, right-click the **graphical execution plan** and click **Save Execution Plan As**.
5. In the **Save As** dialog box, use the file name **D:\Labfiles\Lab07\plan4.sqlplan** and click **Save**.
6. Notice that the relative cost of the first query in the batch has reduced to approximately 20 percent. Also, notice that a seek of the new index is being used, and that the data type conversion warning no longer appears.
7. Close SSMS without saving any changes.

Results: At the end of this lab, you will have examined a query execution plan for a stored procedure and implemented performance improvements by adding a covering index and eliminating an implicit data type conversion.

Lab 8: Plan Caching and Recompilation

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler, and retailer of cycle products. The owners of the company have decided to start a new direct marketing arm. It has been created as a new company named Proseware Inc. Even though it has been set up as a separate company, it will receive some IT-related services from Adventure Works and will be provided with a subset of the corporate Adventure Works data. The existing Adventure Works SQL Server platform has been moved to a new server that is capable of supporting both the existing workload and the workload from the new company.

Objectives

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

- Use the plan cache to identify and resolve query performance issues.
- Use the Query Store to monitor performance, and to force a query plan.

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Troubleshooting with the Plan Cache

Scenario

Since a new Proseware Inc. application started to interact with the **AdventureWorks** database, you have noticed that the plan cache of the SQL Server instance occupies more memory than before. You will investigate this issue by examining the contents of the plan cache, and try to identify a resolution.

You know that the new Proseware Inc. application interacts with the **AdventureWorks** database using stored procedures.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Start the Workload

3. Check for Plan Cache Bloat

4. Identify the Query Causing Plan Cache Bloat

5. Identify the Stored Procedure Causing Plan Cache Bloat

6. Rewrite Proseware.up_CampaignReport to Prevent Plan Cache Bloat

7. Verify That the Stored Procedure Is Using a Single Query Plan

8. Stop the Workload

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

2. Run **Setup.cmd** in the **D:\Labfiles\Lab08\Starter** folder as Administrator.

▶ Task 2: Start the Workload

- In the **D:\Labfiles\Lab08\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell. If a message is displayed asking you to confirm a change in execution policy, type **Y**. Once the workload script is running, continue with the exercise.

▶ Task 3: Check for Plan Cache Bloat

1. Start **SQL Server Management Studio**, then open the project file D:\Labfiles\Lab08\Starter\Project\Project.ssmssln and the Transact-SQL file **Lab Exercise 01 - plan cache.sql**.
2. Under the comment that begins **Task 2**, execute the query against the **sys.dm_exec_query_stats** system DMV to find the most common **query_hash** executed on the **MIA-SQL** instance.

   Is there any indication of plan cache bloat?

▶ Task 4: Identify the Query Causing Plan Cache Bloat

1. Under the comment that begins **Task 3**, edit the query to return an example **plan_handle** from **sys.dm_exec_query_stats** related to the **query_hash** returned by the previous task. To do this, you must replace the text "<query has from task 1>" with the value of the **query_hash** column returned from task 2.

▶ Task 5: Identify the Stored Procedure Causing Plan Cache Bloat

1. Under the comment that begins **Task 4**, execute the query against **INFORMATION_SCHEMA.ROUTINES** to find the stored procedure that is causing plan cache bloat (**INFORMATION_SCHEMA.ROUTINES** contains the code for database objects).
2. Which stored procedure appears to be the best candidate for causing plan cache bloat?
3. Based on the stored procedure code, can you suggest how you might rewrite the procedure to avoid plan cache bloat?

▶ Task 6: Rewrite Proseware.up_CampaignReport to Prevent Plan Cache Bloat

1. From Solution Explorer, open the query file **Lab Exercise 01a -Proseware.up_CampaignReport.sql**. This file contains the definition of the stored procedure.
2. Change the stored procedure definition to remove the use of dynamic SQL.

▶ Task 7: Verify That the Stored Procedure Is Using a Single Query Plan

1. In SSMS, return to the query window where **Lab Exercise 01 - plan cache.sql** is open.
2. Under the comment that begins **Task 6**, execute the query against **sys.dm_exec_procedure_stats** to show the query plan for **Proseware.up_CampaignReport**.
3. Notice that only one row is returned by the query; this indicates that the stored procedure is using only one query plan.

▶ Task 8: Stop the Workload

1. Highlight the code under the comment that begins **Task 7** and execute it.
2. Press ENTER in the PowerShell workload window to close it.

Results: At the end of this exercise, you will have refactored a stored procedure to reduce plan cache bloat.

Exercise 2: Working with the Query Store

Scenario

Some of the data required by Proseware Inc. applications has been added to a new database called **Proseware**. You must configure the Query Store for the new **Proseware** database.

The main tasks for this exercise are as follows:

1. Start the Workload
2. Enable the Query Store
3. Amend the Query Store Statistics Collection Interval
4. Check the Top Resource Consuming Queries Report
5. Add a Missing Index
6. Force a Query Plan
7. Stop the Workload

▶ Task 1: Start the Workload

- In the **D:\Labfiles\Lab08\Starter** folder, execute **start_load_exercise_02.ps1** with PowerShell. Wait a few minutes before continuing.

▶ Task 2: Enable the Query Store

- Use the SSMS GUI to turn on Query Store for the **ProseWare** database.

▶ Task 3: Amend the Query Store Statistics Collection Interval

- Use the SSMS GUI to change the Query Store Statistics Collection Interval to **1 minute**.

▶ Task 4: Check the Top Resource Consuming Queries Report

1. In SSMS, open the Query Store Top Resource Consuming Queries Report for the **ProseWare** database.
2. Note the **query id** of the top query.

▶ Task 5: Add a Missing Index

1. If it is not already open, open the Transact-SQL file **Lab Exercise 02 - Query Store.sql**.
2. Execute the code under **task 5** to create a missing index.

▶ Task 6: Force a Query Plan

1. Open the Query Store Tracked Queries report, and view the details for the query ID you noted in an earlier task.
2. Select one of the query plans for this query and force it.

▶ Task 7: Stop the Workload

1. Return to the query window where **Lab Exercise 02 - Query Store.sql** is open.
2. Execute the code under **Task 7** to stop the workload.
3. Close SQL Server Management Studio without saving any changes.
4. Close the PowerShell workload window.

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

Configure the Query Store.

Use the Query Store to investigate statement query execution plans.

Use the Query Store to force a query execution plan.

Lab Answer Key 8: Plan Caching and Recompilation

Exercise 1: Troubleshooting with the Plan Cache

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. In the **D:\Labfiles\Lab08\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Start the Workload

1. Open Windows Explorer and browse to **D:\Labfiles\Lab08\Starter**.
2. Right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**.
3. If a message is displayed asking you to confirm a change in execution policy, type **Y**, and then press ENTER.
   Once the workload script is running, continue with the exercise. Do not wait for it to finish.

▶ Task 3: Check for Plan Cache Bloat

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab08\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click **Lab Exercise 01 - plan cache.sql**. (If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**.)
5. Select the query under the comment that begins **Task 2**, and then click **Execute**.

   Plan cache bloat is occurring; a single query hash is linked to hundreds of cached plans. Queries that are identical, other than literal values, are assigned the same query hash.

▶ Task 4: Identify the Query Causing Plan Cache Bloat

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

   ```
   SELECT TOP(1) [text]
   FROM sys.dm_exec_query_stats AS qs
   CROSS APPLY sys.dm_exec_sql_text(qs.plan_handle) AS st
   WHERE query_hash = <query hash from task 1>
   ```

   Replace the text "<query hash from task 1>" with the value of the **query_hash** column returned from task 2.
2. Select the query you have amended and click **Execute**.

▶ Task 5: Identify the Stored Procedure Causing Plan Cache Bloat

1. Select the query under the comment that begins **Task 4**, and click **Execute.**
2. The procedure **Proseware.up_CampaignReport** is the only candidate procedure identified by your query.
3. **Proseware.up_CampaignReport** uses a dynamic SQL query; this is the cause of plan cache bloat, because each execution of the dynamic SQL query gets its own query execution plan added to the cache. In this case, the dynamic SQL query is unnecessary and can be removed.

▶ Task 6: Rewrite Proseware.up_CampaignReport to Prevent Plan Cache Bloat

1. In Solution Explorer, double-click the query **Lab Exercise 01a -Proseware.up_CampaignReport.sql**. (If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**.)
2. Amend the stored procedure definition in the file so that it reads:

```
ALTER PROCEDURE Proseware.up_CampaignReport
(@CampaignName varchar(20))
AS
   SELECT  cn.CampaignID,
           cn.CampaignName,
   cn.CampaignStartDate,
   cn.CampaignEndDate,
   st.Name,
   cr.ResponseDate,
   cr.ConvertedToSale,
   cr.ConvertedSaleValueUSD
   FROM Proseware.Campaign AS cn
   JOIN Sales.SalesTerritory AS st
   ON st.TerritoryID = cn.CampaignTerritoryID
   JOIN Proseware.CampaignResponse AS cr
   ON cr.CampaignID = cn.CampaignID
   WHERE CampaignName = @CampaignName;
GO
```

3. Click **Execute.**

▶ Task 7: Verify That the Stored Procedure Is Using a Single Query Plan

1. In the **Lab Exercise 01 - plan cache.sql** pane, select the query under the comment that begins **Task 6** and click **Execute**.
2. Notice that only one row is returned by the query; this indicates that the stored procedure is using only one query plan.

▶ Task 8: Stop the Workload

1. In the query pane, highlight the code under the comment that begins **Task 7** and click **Execute**. This will stop the workload.
2. Press ENTER in the PowerShell workload window to close it.
3. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, you will have refactored a stored procedure to reduce plan cache bloat.

Exercise 2: Working with the Query Store

▶ Task 1: Start the Workload

1. Open Windows Explorer and browse to **D:\Labfiles\Lab08\Starter**.
2. Right-click **start_load_exercise_02.ps1**, and then click **Run with PowerShell**. Allow this to run for a few minutes before continuing.

▶ Task 2: Enable the Query Store

1. In SQL Server Management Studio, in the Object Explorer pane, expand **Databases**, right-click **ProseWare**, and then click **Properties**.
2. In the **Database Properties - ProseWare** dialog box, on the **Query Store** page, change the value of the **Operation Mode (Requested)** property to **Read Write**, and then click **OK**.

▶ Task 3: Amend the Query Store Statistics Collection Interval

1. In SQL Server Management Studio, in the Object Explorer pane, expand **Databases**, right-click **ProseWare**, and then click **Properties**.
2. In the **Database Properties - ProseWare** dialog box, on the **Query Store** page, change the value of the **Statistics Collection Interval** property to **1 minute**, and then click **OK**.

▶ Task 4: Check the Top Resource Consuming Queries Report

1. In Object Explorer, expand **ProseWare**, and then expand **Query Store**.
2. Double-click **Top Resource Consuming Queries**.
3. Hover over the largest bar in the histogram in the upper left of the Top Resource Consumers window, and note the value of **query id**. This is the **query id** of the most expensive query.

▶ Task 5: Add a Missing Index

1. In Solution Explorer, double-click **Lab Exercise 02 - Query Store.sql**.
2. Highlight the code under the comment that begins **task 5**, and click **Execute**.

▶ Task 6: Force a Query Plan

1. In Object Explorer, double-click **Tracked Queries**.
2. In the Tracked Queries pane, in the **Tracking Query** box, type the query id you noted in an earlier task, then press ENTER.
3. In the graph in the upper half of the Tracked Queries pane, click on either of the two points.
4. On the toolbar, click **Force Plan**, and then in the **Confirmation** dialog box, click **Yes**.

▶ Task 7: Stop the Workload

1. Return to the query window where **Lab Exercise 02 - Query Store.sql** is open.
2. In the query pane, highlight the code under the comment that begins **Task 7** and click **Execute**. This will stop the workload.
3. Close SQL Server Management Studio without saving any changes.
4. Close the PowerShell workload window.

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

Configure the Query Store.

Use the Query Store to investigate statement query execution plans.

Use the Query Store to force a query execution plan.

Lab 9: Extended Events

Scenario

While investigating why the new SQL Server instance was so slow, you came across deadlock occurrences and excessive fragmentation in indexes caused by page split operations. In this lab, you will review deadlock occurrences using the default session and implement a new Extended Event session to identify workloads that cause huge page splits.

Objectives

After completing this lab, you will be able to:

- Access data captured by the System Health Extended Events session.
- Create a custom Extended Events session.

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Using the system_health Extended Events Session

Scenario

While investigating why the new SQL Server instance was so slow, you were informed that users frequently report deadlock error messages in application logs. In this exercise, you will review Extended Events default system_health session and analyze the output of a deadlock event.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Run a Workload

3. Query the system_health Extended Events Session

4. Extract Deadlock Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab09\Starter folder** as Administrator.

▶ Task 2: Run a Workload

1. In the **D:\Labfiles\Lab09\Starter** folder, run **start_load_1.ps1** with Windows PowerShell. If a message is displayed asking you to confirm a change in execution policy, type **Y**.
2. Wait for the workload to complete. This should take about 60 seconds.

▶ Task 3: Query the system_health Extended Events Session

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** instance, then **D:\Labfiles\Lab09\Starter\Project\Project.ssmssln** and **Exercise 01 - system_health.sql**.
2. Under the comment that begins **-- Task 2**, edit and execute the query to return data from the system_health session, using the **sys.fn_xe_file_target_read_file** DMF to extract data from the session's event file target.
3. Hint: you can examine the definition of the system_health session to find the file name used by the event file target.

▶ Task 4: Extract Deadlock Data

1. Under the comment that begins **-- Task 3**, edit and execute the query to extract the events with a name attribute of **xml_deadlock_report** from the XML version of the **event_data** column. Include the event time and the **/event/data/value/deadlock** element in your output.
2. Click on any of the row values in the **deadlock_data** column to view the deadlock XML in detail.

Results: After completing this exercise, you will have extracted deadlock data from the SQL Server.

Exercise 2: Tracking Page Splits Using Extended Events

Scenario

While investigating why the new SQL Server instance was so slow, you came across excessive fragmentation in indexes caused by page split operations. In this exercise, you will implement Extended Events to identify those workloads that cause huge page splits.

Note: Although a **page_split** event is available, it doesn't provide enough information for you to discriminate between expected page splits (which occur when a table or index is extended, referred to as *end page splits*) and page splits which can harm performance (which occur when data must be inserted in the middle of a page, referred to as *mid-page splits*). You can detect mid-page splits by analyzing the **transaction_log** event.

The main tasks for this exercise are as follows:

1. Create an Extended Events Session to Track Page Splits
2. Run a Workload
3. Query the Session
4. Extract alloc_unit_id and Count Values
5. Return Object Names
6. Delete the Session

▶ Task 1: Create an Extended Events Session to Track Page Splits

1. In Solution Explorer, open **Exercise 02 – page splits.sql**.
2. Create a new Extended Events session on the **MIA-SQL** instance with the following properties:
   - Session name: **track page splits**
   - Event(s) included: **sqlserver.transaction_log**

- Event filter(s):
  - **operation** = **LOP_DELETE_SPLIT**
  - **database_name** = **AdventureWorks**
- Session target: **Histogram**
  - Filtering target: **sqlserver.transaction_log**
  - Source: **alloc_unit_id**
  - Source type: **event**

▶ Task 2: Run a Workload

1. In the **D:\Labfiles\Lab09\Starter** folder, execute **start_load_2.ps1** with PowerShell.
2. Wait for the workload to complete. This should take about 60 seconds.

▶ Task 3: Query the Session

- In SSMS, in the query window for **Exercise 02 – page splits.sql,** under the comment that begins **-- Task 3**, edit and execute the query to extract data from the histogram target of the **track page splits** session. Use the **sys.dm_xe_session_targets** DMV to extract data from the session's histogram target. For this task, include only the **target_data** column in your output result set and cast the results to XML.

▶ Task 4: Extract alloc_unit_id and Count Values

- Under the comment that begins **-- Task 4**, edit and execute the query so that it returns the **count** attribute for each **HistogramTarget/Slot** node, and the **value** child node for each **HistogramTarget/Slot** node.

▶ Task 5: Return Object Names

- Under the comment that begins **-- Task 5**, edit and execute the query to join to **sys.allocation_units**, **sys.partitions** and **sys.indexes** to find the names of objects affected by page splits.

▶ Task 6: Delete the Session

1. Delete the **track page splits** session.
2. Close any open applications and windows.

Results: After completing this exercise, you will have extracted page split data from SQL Server.

Lab Answer Key 9: Extended Events

Exercise 1: Using the system_health Extended Events Session

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab09\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Run a Workload

1. Open File Explorer and navigate to the **D:\Labfiles\Lab09\Starter** folder.
2. Right-click **start_load_1.ps1** and then click **Run with PowerShell**.
3. If a message is displayed asking you to confirm a change in execution policy, type **Y**, and then press **ENTER**.
4. Wait for the workload to complete—this should take about a minute—and then press **ENTER** to close the Windows PowerShell window.

▶ Task 3: Query the system_health Extended Events Session

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab09\Starter\Project** folder, click **Project.ssmssln**, and then click **Open**.
4. In Solution Explorer, double-click **Exercise 01 - system_health.sql**.
5. Edit the code under the comment that begins **-- Task 2** so that it reads as follows:

```
SELECT CAST(event_data AS xml) AS xe_data
FROM sys.fn_xe_file_target_read_file('system_health*.xel', NULL, NULL, NULL);
```

6. Select the query that you edited in the previous step, and then click **Execute**.

▶ Task 4: Extract Deadlock Data

1. Edit the query under the comment that begins **-- Task 3** so that it reads as follows:

```
SELECT xe_event.c.value('@timestamp', 'datetime2(3)') AS event_time,
xe_event.c.query('/event/data/value/deadlock') AS deadlock_data
FROM
(
   SELECT CAST(event_data AS xml) AS xe_data
   FROM sys.fn_xe_file_target_read_file('system_health*.xel', NULL, NULL, NULL)
) AS xe_data
CROSS APPLY xe_data.nodes('/event') AS xe_event(c)
WHERE xe_event.c.value('@name', 'varchar(100)') = 'xml_deadlock_report'
ORDER BY event_time;
```

2. Select the query that you edited in the previous step, and then click **Execute**.
3. In the Results pane, click on any of the row values in the **deadlock_data** column to view the deadlock XML in detail.
4. Leave SSMS open for the next exercise.

Results: After completing this exercise, you will have extracted deadlock data from the SQL Server.

Exercise 2: Tracking Page Splits Using Extended Events

▶ Task 1: Create an Extended Events Session to Track Page Splits

1. In Solution Explorer, double-click **Exercise 02 – page splits.sql**.
2. In Object Explorer, under **MIA-SQL (SQL Server 13.0.1000 - ADVENTUREWORKS\Student)**, expand **Management**, expand **Extended Events**, right-click **Sessions**, and then click **New Session Wizard**.
3. In the **New Session Wizard**, on the **Introduction** page, click **Next**.
4. On the **Set Session Properties** page, in the **Session name** box, type **track page splits**, and then click **Next**.
5. On the **Choose Template** page, click **Next**.
6. On the **Select Events To Capture** page, in the **Event library** section, click the drop-down button in the **Channel** column header (you may have to scroll right), then select **Debug**. In the first **Search Events** box, type **transaction_log,** double-click the **transaction_log** row in the **Event Library** list, which will add it to the **Selected events** list, and then click **Next**.
7. On the **Capture Global Fields** page, click **Next**.
8. On the **Set Session Event Filters** page, click **Click here to add a clause**. In the **Field** drop-down list, click **sqlserver.database_name**, in the **Value** box, type **AdventureWorks**, and then click **Finish**.
9. On the New Session Wizard: **Create Event Session** page, click **Close.**
10. In Object Explorer, expand **Sessions**, right-click **track page splits**, and then click **Properties**.
11. In the **Session Properties** dialog box, on the **Events** page, click **Configure**.
12. In the **Selected events** list, click **transaction_log**, on the **Filter (Predicate)** tab, click **Click here to add a clause**. Ensure that the value of the **And/Or** box is **And**, in the **Field** list, click **operation**, in the **Operator** list, click **=**, and then in the **Value** list, click **LOP_DELETE_SPLIT**.
13. On the **Data Storage** page, click **Click here to add a target**. In the **Type** list, click **histogram**.
14. In the **Event to filter on** list, click **transaction_log**, in the **Base buckets on** section, click **Field**, in the **Field** list, click **alloc_unit_id**, and then click **OK**.
15. In Object Explorer, right-click **track page splits** and click **Start session**.

▶ Task 2: Run a Workload

1. In File Explorer, navigate to the **D:\Labfiles\Lab09\Starter** folder.
2. Right-click **start_load_2.ps1**, and then click **Run with PowerShell**.
3. Wait for the workload to complete. This should take about 60 seconds.

▶ Task 3: Query the Session

1. In SQL Server Management Studio, in the query window for **Exercise 02 – page splits.sql**, edit the code under the comment that begins **-- Task 3** so that it reads as follows:

```
USE AdventureWorks;
GO
SELECT CAST(target_data AS XML) AS target_data
FROM sys.dm_xe_sessions AS xs
JOIN sys.dm_xe_session_targets xt
ON xs.address = xt.event_session_address
WHERE xs.name = 'track page splits'
AND xt.target_name = 'histogram';
```

2. Select the query that you edited in the previous step, and then click **Execute**.

3. In the results pane, click the returned XML to review the data.

▶ Task 4: Extract alloc_unit_id and Count Values

1. In the Exercise 02 – page splits.sql pane, edit the code under the comment that begins **-- Task 4** so that it reads as follows:

```
SELECT xe_node.value('(value)[1]', 'bigint') AS alloc_unit_id,
xe_node.value('(@count)[1]', 'bigint') AS split_count
FROM (     SELECT CAST(target_data AS XML) AS target_data
FROM sys.dm_xe_sessions AS xs
JOIN sys.dm_xe_session_targets xt
ON xs.address = xt.event_session_address
WHERE xs.name = 'track page splits'
AND xt.target_name = 'histogram')
AS xe_data
CROSS APPLY target_data.nodes('HistogramTarget/Slot') AS xe_xml (xe_node);
```

2. Select the query that you edited in the previous step, and then click **Execute**.

3. Review the number of splits in each node.

▶ Task 5: Return Object Names

1. Edit the code under the comment that begins **-- Task 5** so that it reads as follows:

```
SELECT OBJECT_SCHEMA_NAME(sp.object_id) AS object_schema,
          OBJECT_NAME(sp.object_id) AS object_name,
          si.name AS index_name,
          xe.split_count
FROM (    SELECT xe_node.value('(value)[1]', 'bigint') AS alloc_unit_id,
                         xe_node.value('(@count)[1]', 'bigint') AS split_count
          FROM (  SELECT CAST(target_data AS XML) AS target_data
                         FROM sys.dm_xe_sessions AS xs
                         JOIN sys.dm_xe_session_targets xt
                         ON xs.address = xt.event_session_address
                         WHERE xs.name = 'track page splits'
                         AND xt.target_name = 'histogram') AS xe_data
          CROSS APPLY target_data.nodes('HistogramTarget/Slot') AS xe_xml (xe_node))
AS xe
JOIN sys.allocation_units AS sau
ON sau.allocation_unit_id = xe.alloc_unit_id
JOIN sys.partitions AS sp
ON sp.partition_id = sau.container_id
JOIN sys.indexes AS si
ON si.object_id = sp.object_id
AND si.index_id = sp.index_id;
```

2. Select the query that you edited in the previous step, and then click **Execute**.

3. Review the objects affected by page splits.

▶ Task 6: Delete the Session

1. In Object Explorer, under **Sessions**, right-click **track page splits**, and click **Delete**.

2. In the **Delete Object** dialog box, click **OK**.

3. Close SSMS without saving changes.

4. In the Windows PowerShell window, press **ENTER** to close the window.

Results: After completing this exercise, you will have extracted page split data from SQL Server.

Lab 10: Monitoring, Tracing, and Baselining

Scenario

You are investigating why a new SQL Server instance is so slow; users frequently complain that their workloads run very slowly during peak hours of business. In addition, to troubleshoot performance issues in future and take more informed corrective measures, you decide to establish a baseline for SQL Server performance. In this lab, you will set up data collection for analyzing workload during peak business hours and implement a baseline methodology to collect performance data at frequent intervals, so that comparisons can 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: **10987C-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 Events Session

3. Execute Workload

4. Analyze Collected Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 10987C-MIA-DC and 10987C-MIA-SQL virtual machines are running, and then log on to the 10987C-MIA-SQL machine as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab10\Starter** folder, run **Setup.cmd** as an Administrator.

▶ Task 2: Set up an Extended Events 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\Lab10\Starter** folder, the **SetupExtendedEvent.sql** file has a possible solution script.
2. Watch Live Data for the Extended Events session.

▶ Task 3: Execute Workload

1. In the **D:\Labfiles\Lab10\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.
3. Drop the AnalyzeSQLEE Extended Events session.

Results: After completing this exercise, you will have set up an Extended Events session that collects performance data for a workload and analyzed the data.

Exercise 2: Implementing Baseline Methodology

Scenario

You are asked to set up a baseline methodology to collect data that can be used as baseline for comparison if the instance develops performance issues.

The main tasks for this exercise are as follows:

1. Set up Data Collection Scripts
2. Execute Workload
3. Analyze Data

▶ Task 1: Set up Data Collection Scripts

- Create a database named **baseline** by using default settings, and then clear the wait statistics for the database. In the D:\Labfiles\Lab10\Starter\10987-10 folder, the **PrepareScript.sql** Transact-SQL file has a sample solution script.

▶ Task 2: Execute Workload

1. Create a job from the **WaitsCollectorJob.sql** Transact-SQL file in the **D:\Labfiles\Lab10\Starter** folder.
2. Run the **waits_collections** job to collect statistics before and after running the **RunWorkload.cmd** file multiple times.

▶ Task 3: Analyze Data

1. Using the collected waits data, write and execute a query to find the waits for the workload. In the **D:\Labfiles\Lab10\Starter\10987-10** folder, the **WaitBaselineDelta.sql** file has a sample solution script.
2. Using the collected waits data, write and execute a query to find the percentage of waits. In the **D:\Labfiles\Lab10\Starter\10987-10** folder, the **WaitBaselinePercentage.sql** file has a sample solution script.

3. Using the collected waits data, write and execute a query to find the top 10 waits. In the **D:\Labfiles\Lab10\Starter\10987-10** folder, the **WaitBaselineTop10.sql** file has a sample solution script.
4. Close SQL Server Management Studio without saving any changes.
5. Close File Explorer.

Results: After completing this exercise, you will have implemented a baseline for a workload.

Lab Answer Key 10: Monitoring, Tracing, and Baselining

Exercise 1: Collecting and Analyzing Data Using Extended Events

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 10987C-MIA-DC and 10987C-MIA-SQL virtual machines are running, and then log on to the 10987C-MIA-SQL machine as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab10\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. 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: Set up an Extended Events 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\Lab10\Starter** folder, click **10987-10.ssmssln**, and then click **Open**.
4. In Solution Explorer, under **Queries**, double-click **SetupExtendedEvent.sql**, and then on the toolbar, click **Execute**.
5. In Object Explorer, expand **Management**, expand **Extended Events**, and then expand **Sessions**.
6. Right-click **AnalyzeSQLEE**, and then click **Watch Live Data**.

▶ Task 3: Execute Workload

1. In File Explorer, in the **D:\Labfiles\Lab10\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 **AnalyzeSQLEE: Live Data** 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 **AnalyzeSQLEE: Live Data** pane, 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**, and then click **Execute** to remove the Extended Event.
6. Leave SQL Server Management Studio open for the next exercise.

Results: After completing this exercise, you will have set up an Extended Events session that collects performance data for a workload and analyzed the data.

Exercise 2: Implementing Baseline Methodology

▶ Task 1: Set up Data Collection Scripts

1. In SQL Server Management Studio, in Solution Explorer, double-click **PrepareScript.sql**.
2. Examine the contents of the script, and then click **Execute**. The error can be ignored as this just means the database has already been removed.

▶ Task 2: Execute Workload

1. In Solution Explorer, double-click **WaitsCollectorJob.sql**, and then click **Execute**.
2. In Object Explorer, expand **SQL Server Agent**, expand **Jobs**, right-click **waits_collections**, and then click **Start Job at Step**.
3. Wait for the job to complete, and then click **Close**.
4. In File Explorer, navigate to the **D:\Labfiles\Lab10\Starter** folder, right-click **RunWorkload.cmd**, and then click **Run as administrator**.
5. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.
6. In SQL Server Management Studio, in **Jobs**, right-click **waits_collections**, and then click **Start Job at Step**.
7. Wait for the job to complete, and then click **Close**.

▶ Task 3: Analyze Data

1. In Solution Explorer, double-click **WaitBaselineDelta.sql**, and then click **Execute**.
2. In Solution Explorer, double-click **WaitBaselinePercentage.sql**, and then click **Execute**.
3. In Solution Explorer, double-click **WaitBaselineTop10.sql**, and then click **Execute**.
4. In the Results pane, observe the top 10 waits that were collected during the execution of the workload.
5. Close SQL Server Management Studio without saving any changes.
6. Close File Explorer.

Results: After completing this exercise, you will have implemented a baseline for a workload.