Module 2: Table and Index Structure

Student Lab Manual

SQL Server 2012: Performance Tuning – Design, Internals, and Architecture

Version 1.0

Conditions and Terms of Use

Microsoft Confidential - For Internal Use Only

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at
http://www.microsoft.com/about/legal/permissions/

Microsoft®, Internet Explorer®, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Schemas

Introduction

This lab will provide you hands-on experience with Schemas and DMV's.

Objectives

After completing this lab, you will be able to:

- Identify how many Indexes and what types, exists in a particular table.
- Provide a list of partitions if available.
- Familiarize with execution plans index operators .

Prerequisites

You should have completed Lesson 1 and 2 of the Table and Index Structure Module.

Estimated time to complete this lab

40 minutes

Scenario

There is a requirement to provide a list of the different index types in some tables, and a list of partitions (if exists), this in order to analyze the existing indexes so new indexes can be created as needed.

Exercise 1: Supporting DMV´s

Objectives

In this exercise, you will:

- Learn about the relationship between sys.objects, sys.partitions, and sys.indexes.
- Observe INDID = 0 is a heap, INDID=1 table has a clustered index
- Observe simple ShowPlans.

All scripts for this lab are located in the C:\Labs\Module2\Exercise1 folder.

Prerequisites

- Connect to the SQL2012PT Virtual Machine
- Log in using the following credentials
  User: Administrator
  Password: P@ssw0rd

> **Note:** The Virtual Machine for this workshop is time bombed for security purposes. You may need to rearm the virtual machine if the activation has expired. If the VM issues a message that it needs to be reactivated, you can use slmgr.vbs with the rearm option as follows:
>
> 1. Open an elevated command prompt (right click on “Command Prompt” in the Start menu and click “Run As Administrator)
> 2. Execute the following command
>   *slmgr.vbs –rearm*

Observe indexes

1. Open and execute CreateNewPerson.sql
2. Compare the differences between the indexes for the original Person.Person table and the NewPerson table by running the code below which can be found in the Exercise1_Step2.sql script:

```
Use AdventureWorksPTO
go

select * from sys.objects where  name= 'Person'
select * from sys.indexes where object_id =
   object_id('Person.Person')
select * from sys.partitions where object_id =
   object_id('Person.Person')
select * From sys.stats where object_id =
   object_id('Person.Person')
go

select * from sys.objects where  name= 'NewPerson'
```

```
select * from sys.indexes where object_id =
   object_id('Person.NewPerson')
select * from sys.partitions where object_id =
   object_id('Person.NewPerson')
select * From sys.stats where object_id =
   object_id('Person.NewPerson')
go
```

Notice that the object_id in sys.objects is the object_id in sys.indexes.

Question A: How many indexes are in Person.Person?

Question B: Which one has INDEX_ID = 1?

Question C: Is this a clustered or nonclustered index?

Question D: How many indexes are in NewPerson?

Question E: The script did not create any indexes for this new table. Considering this, what does INDEX_ID=0 mean for NewPerson?

3. Execute a query against a table with no indexes (a HEAP table) and click the hyperlink for the Showplan at the end of your result set to see the execution plan. Also click the messages tab and note the logical and physical reads that are necessary for this. The query below can be found in the Exercise1_Step3.sql script.

> Note: In SQL 2005 when clicking a Showplan link you will be taken to the XML code view of the execution plan. Since SQL 2008 this has changed so that it takes you to the graphical view by default.

```
USE [AdventureWorksPTO]
GO

SET STATISTICS XML ON
SET STATISTICS TIME ON
SET STATISTICS IO ON
GO

SELECT LastName
FROM Person.NewPerson
WHERE LastName BETWEEN 'Devon' AND 'Mendel'
GO

SET STATISTICS XML OFF
SET STATISTICS TIME OFF
SET STATISTICS IO OFF
GO
```

The important point to note here is that a Table Scan is required to find the matching rows, because there are no indexes on the NewPerson table.

4. Execute the following (found in the Exercise1_Step4.sql script) and compare the entries for NewPerson in the system tables:

```
USE [AdventureWorksPTO]
GO

SELECT * FROM sys.objects WHERE  name= 'NewPerson'
SELECT * FROM sys.indexes WHERE object_id =
   object_id('Person.NewPerson')
SELECT * FROM sys.partitions WHERE object_id =
   object_id('Person.NewPerson')
SELECT * FROM sys.stats WHERE object_id =
   object_id('Person.NewPerson')
GO
```

Note sys.stats now contains a row. These stats were generated automatically as a result of the query run in step 3.

5. Contrast the execution plan when an index is added to the NewPerson table. This query can be found in the Exercise1_Step5.sql script.

```
USE [AdventureWorksPTO]
GO

CREATE INDEX idx_Last_Name ON Person.NewPerson (LastName)
```

```
GO

SET STATISTICS XML ON
SET STATISTICS TIME ON
SET STATISTICS IO ON
GO

SELECT LastName
FROM Person.NewPerson
WHERE LastName BETWEEN 'Devon' AND 'Mendel'
GO

SET STATISTICS XML OFF
SET STATISTICS TIME OFF
SET STATISTICS IO OFF
GO
```

Note the change in the execution plan. Also notice the number of logical and physical reads versus the logical and physical reads for a table scan. You find these on the messages tab of your result set.

6. Execute the query below from Exercise1_Step6.sql and look at the execution plan

```
USE [AdventureWorksPTO]
GO

SET  SHOWPLAN_XML ON
GO

SELECT FirstName, LastName
FROM Person.NewPerson
WHERE FirstName = 'Devon'
GO

SET  SHOWPLAN_XML OFF
GO
```

Question F: Is there any difference between this query and the previous one?

Question G: Explain why or why not:

Execute the query below from the same script file to create a new non-clustered index on the Person.NewPerson table and then re-run the select statement from above:

```
-- Composite Index
USE [AdventureWorksPTO]
GO

IF EXISTS (SELECT * FROM sys.indexes WHERE name = 'idx_First_Name'
          and object_id = object_id('Person.NewPerson'))
   DROP INDEX Person.NewPerson.idx_First_Name
GO

CREATE INDEX idx_First_Name on Person.NewPerson(FirstName, LastName)
go

SET  SHOWPLAN_XML ON
GO

SELECT FirstName, LastName
FROM Person.NewPerson
WHERE FirstName = 'Devon'
GO

SET  SHOWPLAN_XML OFF
GO
```

Question H: What is different about this query from the previous one?

Now execute the following query from the same script file which will drop the index you just created and create a new index using the INCLUDE clause and then re-run the select statement from above:

```
-- Covered query using the Include Clause
USE [AdventureWorksPTO]
GO

IF EXISTS (SELECT * FROM sys.indexes WHERE name = 'idx_First_Name'
          and object_id = object_id('Person.NewPerson'))
   DROP INDEX Person.NewPerson.idx_First_Name
GO

CREATE INDEX idx_First_Name on Person.NewPerson(FirstName)
   INCLUDE (LastName)
go

SET  SHOWPLAN_XML ON
GO

SELECT FirstName, LastName
```

```
FROM Person.NewPerson
WHERE FirstName = 'Devon'
GO

SET  SHOWPLAN_XML OFF
GO
```

Question I: What is different about this query from the previous one?

7. Now create a clustered index with the nonclustered index and see which index the optimizer chooses.

```
USE [AdventureWorksPTO]
GO

IF EXISTS (SELECT * FROM sys.indexes WHERE name = 'idx_First_Name'
          and object_id = object_id('Person.NewPerson'))
   DROP INDEX Person.NewPerson.idx_First_Name
GO

IF EXISTS (SELECT * FROM sys.indexes WHERE name = 'idx_Last_Name'
          and object_id = object_id('Person.NewPerson'))
   DROP INDEX Person.NewPerson.idx_Last_Name
GO

CREATE INDEX idx_First_Name on Person.NewPerson(FirstName)
go

CREATE CLUSTERED INDEX idx_Last_Name on Person.NewPerson(LastName)
go

SET  SHOWPLAN_XML ON
GO

SELECT FirstName, LastName
FROM Person.NewPerson
WHERE FirstName = 'Devon'
GO

SET  SHOWPLAN_XML OFF
GO
```

Question J: Was the last query covered or not?

Columnstore Indexes

Introduction

This lab will provide you hands-on experience with Columnstore indexes.

Objectives

After completing this lab, you will be able to:

- Identify when to use a Columnstore Index.

Prerequisites

To have completed Lesson 2 of the Table and Index Structure Module.

Estimated time to complete this lab

20 minutes

Scenario

There is a query that is not performing very well in the Data Warehouse environment; a lot of I/O is consumed by this query and you have to fix it.

Exercise 2: Columnstore indexes

Objectives

In this exercise, you will:

- Observe how a Columnstore index works and how fast it is.

All scripts for this lab are located in the C:\Labs\Module2\Exercise2 folder.

Columnstore vs. Clustered index

1) Open Step1_Create_TableWithoutColumStoreIndex.sql and execute the script. This script may take several minutes to complete, might be a good time to go grab a cup of coffee.

```
Use AdventureWorksPTO
go

select * into dbo.TableWithoutColumnStoreIndex
from Sales.SalesOrderDetail
go

declare @count int =0
while @count < 6
begin
	insert into dbo.TableWithoutColumnStoreIndex
		([SalesOrderID],[CarrierTrackingNumber],[OrderQty],
			[ProductID],[SpecialOfferID],[UnitPrice],
			[UnitPriceDiscount],[LineTotal],rowguid,
			[ModifiedDate])
	select [SalesOrderID],[CarrierTrackingNumber],[OrderQty],
		[ProductID],[SpecialOfferID],[UnitPrice],
		[UnitPriceDiscount],[LineTotal],NEWID(), [ModifiedDate]
	from dbo.TableWithoutColumnStoreIndex

	set @count=@count+1
end

CREATE CLUSTERED INDEX [IDX_Clustered] ON
[dbo].[TableWithoutColumnStoreIndex]
(
	[SalesOrderDetailID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
	SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
	ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO

CREATE NONCLUSTERED INDEX [IDX_NonClustered] ON
[dbo].[TableWithoutColumnStoreIndex]
(
	[SalesOrderID] ASC,
```

```
	[ProductID] ASC
)
INCLUDE (	[ModifiedDate]) WITH (PAD_INDEX = OFF,
STATISTICS_NORECOMPUTE = OFF,
	SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
	ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
```

2) Open Step2_Create_TableWithColumStoreIndex.sql and execute the script. This script may also take some time to complete, how about a snack to go with that coffee?

```
use AdventureWorksPTO
go

select * into dbo.TableWithColumnStoreIndex
from Sales.SalesOrderDetail where 1=2
go

Alter Table dbo.TableWithColumnStoreIndex
Alter Column LineTotal numeric(18,6)
go

insert into dbo.TableWithColumnStoreIndex
	([SalesOrderID],[CarrierTrackingNumber],[OrderQty],
	[ProductID],[SpecialOfferID],[UnitPrice],
	[UnitPriceDiscount],[LineTotal],rowguid, [ModifiedDate])
select [SalesOrderID],[CarrierTrackingNumber],[OrderQty],
	[ProductID],[SpecialOfferID],[UnitPrice],
	[UnitPriceDiscount],[LineTotal],rowguid, [ModifiedDate]
from Sales.SalesOrderDetail
go

declare @count int =0
while @count < 6
begin
	insert into dbo.TableWithColumnStoreIndex
		([SalesOrderID],[CarrierTrackingNumber],
		[OrderQty],[ProductID],
		[SpecialOfferID],[UnitPrice],
		[UnitPriceDiscount],[LineTotal],
		rowguid, [ModifiedDate])
	select [SalesOrderID],[CarrierTrackingNumber],[OrderQty],
		[ProductID],[SpecialOfferID],[UnitPrice],
		[UnitPriceDiscount],[LineTotal],
		NEWID(), [ModifiedDate]
	from dbo.TableWithColumnStoreIndex

	set @count=@count+1
end


Create NonClustered ColumnStore Index IDX_ColumnStore
on dbo.TableWithColumnStoreIndex
	(	[SalesOrderID],
		[SalesOrderDetailID],
		[CarrierTrackingNumber],
```

```
        [OrderQty],
        [ProductID],
        [SpecialOfferID],
        [UnitPrice],
        [UnitPriceDiscount],
        [LineTotal],
        [ModifiedDate]
    )
```

3) Use catalog views to answer the following questions about the tables created above. Some sample queries can be found in the Step3_MetadataQueries.sql script file. You may want to open Books Online to read more about the various catalog views available.

Questions 1: How many indexes are there on the table dbo.TableWithoutColumnStoreIndex?

Question 2: What's the total number of records in the table?

Question 3: What are the indexes on the table dbo.TableWithColumStoreIndex?

Question 4: Which columns is this index created on?

4) Open the script Step4_SelectQuery1.sql and execute the first select query in the script which runs against the table without the columnstore index. Examine the query plan and note down the execution time from the statistics time output on the Messages tab.

```
use AdventureWorksPTO
go
```

```
/******** Query1: Simple Aggregate Query on One table without a
Columnstore index ********/

set statistics time on
go
set statistics xml on
go

select ProductID, Sum(OrderQty) as OrderedQuantity,
       Avg(UnitPrice) As AvgUnitPrice,
       Avg([UnitPriceDiscount]) as AvgDiscountOnUnitPrice
from [dbo].[TableWithoutColumnStoreIndex]
where [UnitPriceDiscount] <> 0.00
Group by ProductID


set statistics time off
go
set statistics xml off
go
```

Run the second query in the file which is against the table with the columnstore index. Examine the execution plan and note down the execution time from the statistics time output on the Messages tab.

```
use AdventureWorksPTO
go

set statistics time on
go
set statistics xml on
go

/**** Query2: Simple Aggregate Query on One table with a Columnstore
index ****/
select ProductID, Sum(OrderQty) as OrderedQuantity,
       Avg(UnitPrice) As AvgUnitPrice,
       Avg([UnitPriceDiscount]) as AvgDiscountOnUnitPrice
from [dbo].[TableWithColumnStoreIndex]
where [UnitPriceDiscount] <> 0.00
Group by ProductID

set statistics time off
go
set statistics xml off
go
```

Question 5: Which query has a faster execution time? Was the columnstore index helpful?

5) Run the queries in the Step5_SelectQuery2.sql script file individually. Examine the execution plans and note down the execution time from the statistics time output on Messages tab for each query.

```
/******** Query1: Complex Query on Multiple Tables without a
Columnstore index ***********/
use AdventureWorksPTO
go

set statistics time on
go
set statistics profile on
go

select ( ISNull(P.FirstName,'') + ' ' + ISNull(P.MiddleName,'')
              + ' ' + IsNull(P.LastName,'')) As CustomerName,
       SOH.OrderDate, SOH.DueDate, SOH.ShipDate,SOH.TotalDue,
       sum(TWC.OrderQty) As TotalOrderQuantity,
       avg(TWC.UnitPrice) As AvgUnitPrice,
       Avg(TWC.UnitPriceDiscount) as AvgDiscountOnUnitPrice
from [dbo].[TableWithoutColumnStoreIndex] TWC
       join Sales.SalesOrderHeader SOH on TWC.SalesOrderID =
                                          SOH.SalesOrderID
       join Sales.Customer C on SOH.CustomerID = C.CustomerID
       join Person.Person P on P.BusinessEntityID = C.PersonID
where TWC.UnitPriceDiscount <> 0 and TWC.OrderQty > 500
group by ( ISNull(P.FirstName,'') + ' ' + ISNull(P.MiddleName,'')
              + ' ' + IsNull(P.LastName,'')),
       SOH.OrderDate, SOH.DueDate, SOH.ShipDate,SOH.TotalDue

set statistics time off
go
set statistics profile off
go


/**** Query2: Complex Query on Multiple Tables with a Columnstore
index ********/
use AdventureWorksPTO
go

set statistics time on
go
set statistics profile on
go

select ( ISNull(P.FirstName,'') + ' ' + ISNull(P.MiddleName,'')
              + ' ' + IsNull(P.LastName,'')) As CustomerName,
       SOH.OrderDate, SOH.DueDate, SOH.ShipDate,SOH.TotalDue,
       sum(TWC.OrderQty) As TotalOrderQuantity,
       avg(TWC.UnitPrice) As AvgUnitPrice,
       Avg(TWC.UnitPriceDiscount) as AvgDiscountOnUnitPrice
from [dbo].[TableWithColumnStoreIndex] TWC
       join Sales.SalesOrderHeader SOH on TWC.SalesOrderID =
                                          SOH.SalesOrderID
       join Sales.Customer C on SOH.CustomerID = C.CustomerID
```

```
        join Person.Person P on P.BusinessEntityID = C.PersonID
where TWC.UnitPriceDiscount <> 0 and TWC.OrderQty > 500
group by ( ISNull(P.FirstName,'') + ' ' + ISNull(P.MiddleName,'')
              + ' ' + IsNull(P.LastName,'')),
        SOH.OrderDate, SOH.DueDate, SOH.ShipDate,SOH.TotalDue

set statistics time off
go
set statistics profile off
go
```

Question 6: Which query has a faster execution time? Was the columnstore index helpful?

Understanding Index Structure and Data Access

Introduction

This lab will provide you understanding on how indexes work and how data is accessed.

Objectives

After completing this lab, you will be able to:

- Understand how data is accessed through the different combination of indexes.

Prerequisites

To have completed Lesson 3 of the Table and Index Structure Module.

Estimated time to complete this lab

40 minutes

Exercise 3.1: Heap Structure

Objectives

In this exercise, you will:

- Observe the structure of a heap table.

All scripts for this lab are located in the C:\Labs\Module2\Exercise3 folder.

Understanding Heaps

1) Open the Create_Table.sql file and execute the script

```
USE AdventureWorksPTO
GO

--Create table to provide the context
CREATE TABLE [dbo].[IndexStructureTable](
          [Id] [int] IDENTITY(1,1) NOT NULL,
          [FirstName] [nvarchar](50) NOT NULL,
          [MiddleName] [nvarchar](50) NULL,
          [LastName] [nvarchar](50) NOT NULL
    ) ON [PRIMARY]
GO

--Insert data
INSERT INTO [dbo].[IndexStructureTable]
    ([FirstName], [MiddleName], [LastName])
SELECT FirstName, MiddleName, LastName
FROM Person.Person
GO
```

2) Open the Index_Structure_Heap.sql file.
3) Run the first query:

```
Use adventureWorksPTO
GO

--So far, there is only a heap table
--Query the sys.indexes

select *
from sys.indexes
where object_id = object_id('dbo.IndexStructureTable')
GO
```

Question 1: Is there any index? Take note of the object_id.

4) Enable the trace flag that will allow us to view the structure of the pages:

```
--Let´s see the structure
DBCC TRACEON (3604)
GO
```

> **Note:** DBCC PAGE and DBCC IND are not officially documented commands. They are "lightly" documented in that they do appear in some official Microsoft Knowledgebase articles, but you do not want to rely on these commands or their output in any applications. They are simply used for illustration purposes here.

5) Run the next statement:

```
--Use DBCC IND to view the Index structure
--DBCC IND (<'DBNAME'>, <'TableNAME'>, <IndexID: 0 (heap),
--         1 (Clustered Index), 2 to 255 (Non-Clustered Index)>)

DBCC IND ('AdventureWorksPTO', 'dbo.IndexStructureTable', 0)
GO
```

Question 2: How many rows there are? Note that the ObjectID is the same you took note of before. Take note of the first PagePID, is it the same as the IAMPID in subsequent rows? What does this means?

6) Take note of another PagePID.

7) Now you are going to use the DBCC PAGE command, but first we need to know the DB_ID, so run the following statement:

```
--Use DBCC PAGE to see what is on the pages
--DBCC PAGE (<DBID>, <FILEID>, <PAGENUMBER>, <0,1,2,3>)

Select DB_ID()
GO
```

8) Take note of the Database ID

9) Now run the following statement to see the structure of the page. Replace the Database ID with the ID you got and the PagePID with the first PagePID you noted:

```
--Checking the first PagePID
DBCC PAGE (<DatabaseID>,1,<PagePID>,3)
```

Question 3: What type of page you would say you are looking at?

Question 4: Can determine how many allocated pages there are? How many?

10) Now run the same statement changing the PagePID with the second PagePID you noted:

```
--Checking the first PagePID
DBCC PAGE (<DatabaseID>,1,<PagePID>,3)
```

Question 5: Can you identify the page type you are looking at? Which type is it?

Question 6: How many slots are? What does this mean?

Exercise 3.2: Clustered Index Structure

Objectives

In this exercise, you will:

- Observe the structure of a Clustered Indexes.

All scripts for this lab are located in the C:\Labs\Module2\Exercise3 folder.

Understanding Clustered Indexes

1) Open the Index_Structure_Clustered.sql file.
2) Run the first statement of the Exercise 3.2 script to create a non-unique clustered index:

```
Use AdventureWorksPTO
GO
--Let's create a non-unique Clustered Index

CREATE CLUSTERED INDEX [CI_ID] ON [dbo].[IndexStructureTable]
(
	[Id] ASC
)
ON [PRIMARY]
GO
```

3) And now check the sys.indexes catalog view to see what you got:

```
--Let's see what we got
select *
from sys.indexes
where object_id = object_id('dbo.IndexStructureTable')
```

Question 1: Is there any Index? What type? Take note of the object_id.

_____

_____

4) Run the following statement to see the index structure:

```
--Let´s see the Index Structure
DBCC TRACEON (3604)
GO
DBCC IND ('AdventureWorksPTO', 'dbo.IndexStructureTable', 1)
GO
```

Question 2: How many rows there are? Note that the ObjectID is the same as the one you noted. Find the row with IndexLevel = 1 and take note of the PagePID. This will not be the first row of the resultset so you may have scroll down a bit to find it.

5) Run the DBCC PAGE command to see the index pages structure. The Database ID should be the same you used before (Exercise 1) and the PagePID will be the one corresponding to the IndexLevel = 1

```
--Check page with Index_Level=1
   DBCC PAGE (<DatabaseID>,1,<PagePID_IndexLevel=1>,3)
```

Question 3: How many rows there are? If you add 2 to the number of rows returned, how many rows you will have? Is it the same amount of rows that you noted in the DBCC IND command? Can you explain why?

Question 4: Why is there a "UNIQUIFIER (key)" column?

Question 5: Do you recognize the level of the B-Tree you are looking at? Which level is it?

6) Take note of the ChildPageId in the **second** row and the corresponding "Id (key)" column.
7) Run the DBCC PAGE command replacing the PagePID with the ChildPageId

```
--Check for a ChildPageId page
DBCC PAGE (<DatabaseID>,1,<PagePID_ChildPageId>,3)
```

Question 6: How many slots there are? What is the value for Id in the Slot 0? Is it the same as the "Id (key)" value you noted before? Can you explain why or why not?

8) Let's drop the non-unique clustered index and create a unique one by issuing the following statement:

```
--Let's create a Unique Clustered Index now
DROP INDEX [CI_ID] ON [dbo].[IndexStructureTable] WITH ( ONLINE = OFF )
GO
```

```
CREATE UNIQUE CLUSTERED INDEX [CI_ID_Unique] ON [dbo].[IndexStructureTable]
(
	[Id] ASC
)
ON [PRIMARY]
GO
```

9) Check if the new index exists by checking the sys.indexes catalog view:

```
--Let's see what we got
select *
from sys.indexes
where object_id = object_id('dbo.IndexStructureTable')
```

10) See the index structure by issuing the DBCC IND command:

```
DBCC IND ('AdventureWorksPTO', 'dbo.IndexStructureTable', 1)
```

11) Take note of the PagePID of the IndexLevel=1 row.

12) Run the DBCC PAGE command using the same Database ID and for the PagePID use the one you noted:

```
--Check page with Index_Level=1
DBCC PAGE (<DatabaseID>,1,<PagePID_IndexLevel=1>,3)
```

Question 7: Is there a UNIQUIFIER column? Why or why not?

13) Take note of the ChildPageId of the **second** row and the corresponding "Id (key)" column.

14) Run the DBCC PAGE command replacing the PagePID with the ChildPageId>

```
--Check for a ChildPageId page
DBCC PAGE (<DatabaseID>,1,<PagePID_ChildPageId>,3)
```

Question 8: How many slots there are? Compared against the non-unique clustered index, do you see more or less slots now? Can you explain why?

Exercise 3.3: Non-Clustered Index Structure and Data Access

Objectives

In this exercise, you will:

- Observe the structure of a Non-Clustered Index.

All scripts for this exercise are in the C:\Labs\Module2\Exercise3 folder.

Understanding Non-Clustered Indexes

1) Open the Index_Structure_NonClustered.sql file.
2) Run the statements from the script below to drop the existing indexes and create a non-clustered index:

```
AdventureWorksPTO
GO
--Let's create a non-clustered index
DROP INDEX [CI_ID_Unique]
       ON [dbo].[IndexStructureTable] WITH ( ONLINE = OFF )
GO

CREATE NONCLUSTERED INDEX [NCI_FirstName] ON [dbo].[IndexStructureTable]
(
       [FirstName] ASC
)
ON [PRIMARY]
GO
```

3) Check the sys.indexes DMV to see what we now have in place on the table:

```
--Let's see what we got
select *
from sys.indexes
where object_id = object_id('dbo.IndexStructureTable')
```

Question 1: Are there any indexes? What type? Take note of the index_id.

Question 2: Do you see a Heap? Can you explain why or why not?

4) Let´s check the index structure by issuing a DBCC IND command:

```
--Let´s check the Index structure
DBCC TRACEON (3604)
GO

DBCC IND ('AdventureWorksPTO', 'dbo.IndexStructureTable', 2)
```

5) Take note of the PagePID of the IndexLevel=1 row.
6) Run the DBCC PAGE using the same Database ID and replacing the PagePID with the one you noted:

```
--Check page with Index_Level=1
DBCC PAGE (<DatabaseID>,1,<PagePID_IndexLevel=1>,3)
```

Question 4: Why is there a "HEAP RID (key)" column? Take note of the ChildPageId in the **second** row. Note that the KeyHashValue column is empty.

Question 5: Besides the HEAP RID, do you see another Key column? Which one?

7) Take note of the column that appears in the output.
8) Run the DBCC PAGE command with the new ChildPageId as the PagePID:

```
--Check for a ChildPageId page
DBCC PAGE (<DatabaseID>,1,<PagePID_ChildPageId>,3)
```

Question 6: How many rows there are?

Question 7: Do you see any values in the KeyHashValue column? Can you explain why?

9) Let's include some columns on the leaf level of the non-clustered index by issuing the following statements:

```
--Let's INCLUDE some columns to this index
DROP INDEX [NCI_FirstName] ON [dbo].[IndexStructureTable]
GO

CREATE NONCLUSTERED INDEX [NCI_FirstName_WithIncludedcolumns] ON
[dbo].[IndexStructureTable]
(
      [FirstName] ASC
)
INCLUDE ([MiddleName], [LastName])
ON [PRIMARY]
GO
```

10) Check the sys.indexes catalog view:

```
--Let's see what we got
select *
from sys.indexes
where object_id = object_id('dbo.IndexStructureTable')
```

11) There is still the heap and the newly created non-clustered index.

12) Issuing the DBCC IND command to see the index structure:

```
DBCC IND ('AdventureWorksPTO', 'dbo.IndexStructureTable', 2)
```

13) Take note of the PagePID for the IndexLevel = 1 row.

14) Run the DBCC PAGE command changing the PagePID with the value you just noted:

```
--Check page with Index_Level=1
DBCC PAGE (<DatabaseID>,1,<PagePID_IndexLevel=1>,3)
```

15) Take note of the columns. Take note of the ChildPageId of the **second** row and run the DBCC PAGE command again, this time with the new value of ChildPageId:

```
--Check for a ChildPageId page
DBCC PAGE (<DatabaseID>,1,<PagePID_ChildPageId>,3)
```

Question 8: Are there any additional columns compared to the last output of the DBCC PAGE? Which columns? What does it tell you?

16) Now, you are going to create a unique clustered index:

```
--Let's create the unique clustered index now
CREATE UNIQUE CLUSTERED INDEX [CI_ID_Unique] ON [dbo].[IndexStructureTable]
(
	[Id] ASC
)
ON [PRIMARY]
GO
```

17) Check the sys.indexes catalog view again:

```
--Let's see what we got
select *
from sys.indexes
where object_id = object_id('dbo.IndexStructureTable')
--The Heap has dissapeared and the clustered index is there.
```

18) Note that the heap has disappeared, so now you have 2 types of indexes, the clustered index and the non-clustered.

19) Let´s see the structure of the non-clustered index first:

```
DBCC IND ('AdventureWorksPTO', 'dbo.IndexStructureTable', 2)
```

20) Take note of the PagePID for the IndexLevel = 1 row value and run the DBCC PAGE with this PagePID:

```
--Check page with Index_Level=1
DBCC PAGE (<DatabaseID>,1,<PagePID_IndexLevel=1>,3)
```

Question 9: Do you see any new columns compared to when you did not have a clustered index? Compare it with the columns you took note in the step 7. What happened?

21) Take note of the second ChildPageId and run the DBCC PAGE with this value as PagePID:

```
--Check for a ChildPageId page
DBCC PAGE (<DatabaseID>,1,<PagePID_ChildPageId>,3)
```

Question 10: Knowing that you are in the leaf level, what has changed here? How many columns do you now? What does it mean?

Clustered Indexes and Page Splits

Introduction

This lab will provide you Understanding on why page splits occur.

Objectives

After completing this lab, you will be able to:

- Identify when to use a clustered index.

Prerequisites

To have completed Lesson 3 of the Table and Index Structure Module.

Estimated time to complete this lab

30 minutes

Exercise 4: Index Fragmentation and Page Splits

Upon completing this exercise, you will be able to:

- Observe page splitting and its causes
- Use dm_db_index_physical_stats to determine if an index is fragmented
- If Fragmentation is a concern

All scripts for this lab are located in C:\Labs\Module2\Exercise4

Create a page Split scenario

1. Execute *1.CreateTable.sql*.

```
USE AdventureworksPTO
GO

IF  object_id('PgSplit') is not null
BEGIN
   DROP TABLE PgSplit
END
GO

CREATE TABLE PgSplit (col1 int ,col2 char(3950))
GO

CREATE UNIQUE CLUSTERED INDEX PgSplit_ind on PgSplit(col1)
GO
```

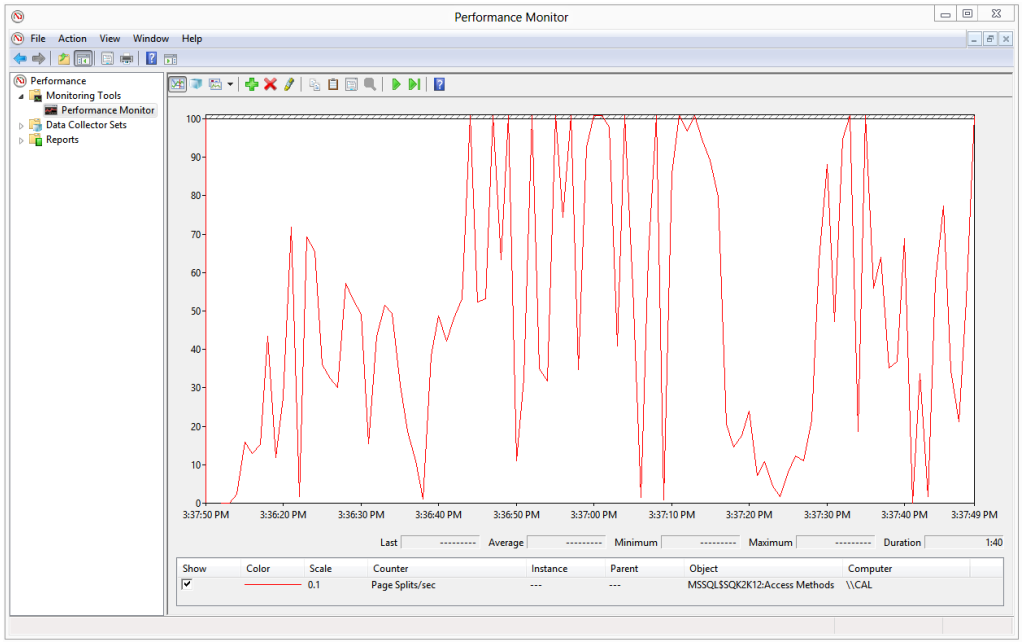
Note the clustered index on col1. This means the rows will be physically ordered on the data pages using this column.

Also, the col2 field is a fixed character column with a width of 3950. Considering that each page in SQL Server is roughly 8K, only two rows will fit on each data page. This was done in order to more easily observe page splits.

Start Performance Monitor and monitor page splits/sec

2. On the Start menu, click Run.
3. Type PERFMON.EXE in the edit box and press ENTER.
4. Click “Performance Monitor” in the left nav pane.
5. Highlight any counters in the bottom pane of perfmon and press the DELETE key to remove them.
6. Click the green PLUS button on the toolbar to add counters.
7. In the listbox under “Available counters,” click the + sign next to SQL Server:Access Methods to expand the list of counters.

8. Select the “Page Splits/sec” performance counter.
9. Click the “Add >>” button to add this counter.
10. Click the “OK” button to return to perfmon.



11. Watch the Page Splits/Sec counter in perfmon, then execute the code in *2.InsertPagesplit.sql*.

```
use AdventureWorksPTO
go

SET NOCOUNT ON
GO

DECLARE @i int
SET @i=0

WHILE (@i<5000)
BEGIN
    INSERT INTO PgSplit VALUES(@i,'___')
    INSERT INTO PgSplit VALUES(10000-@i,'___')
    SET @i=@i+1
END
GO

SET NOCOUNT OFF
GO
```

Notice the loop is appending 10000 rows to the table out of order for the clustered index on col1 (0, 10000, 1, 9999, 2, 9998, 3, 9997, 4, 9996…4999,5000). Inserting the rows out of order for the clustered index forces SQL Server to periodically split a

data page in order to keep the rows physically ordered by the clustered index. Large rows like the ones in this example make for more frequent page splits.

Question K: What was the max page splits/sec in perfmon? Note: Your result may be different than other students doing this and other exercises

12. Execute the following in a New Query window (The queries to run for this step are located in the *3.ShowStatisticsIO.sql* file), note the number of logical reads from the STATISTICS IO output on the Messages tab.

```
Use AdventureworksPTO
Go

SET STATISTICS IO ON
GO

SELECT *
FROM PgSplit

SET STATISTICS IO OFF
GO
```

Question L: At the end of the query results (text results) or on the Messages tab (table results), how many logical reads are needed to service this query?

Observe the amount of fragmentation

13. Execute *4.ExamineFragmentation.sql*.

```
use AdventureworksPTO
go
DECLARE @db_id SMALLINT;
DECLARE @object_id INT;

SET @db_id = DB_ID(N'AdventureWorksPTO');
SET @object_id = OBJECT_ID(N'AdventureWorksPTO.dbo.PgSplit');

SELECT *
FROM sys.dm_db_index_physical_stats(@db_id,
```

```
        @object_id, NULL, NULL , 'LIMITED');
go
```

Question M: What are the values for avg_fragmentation_in_percent and page_count?

Contrast the behavior to rows inserted in clustered –index-order

14. If you closed it, open up Perfmon again and set up and run the counter as above (steps 9-17). Execute the SQL script in *5.PageSplit2.sql*.

```
-- Create Table with Clustered Index
use AdventureWorksPTO
go

IF  object_id('NoPgSplit') is not null
BEGIN
   DROP TABLE NoPgSplit
END
GO

CREATE TABLE NoPgSplit (col1 int ,col2 char(3950))
GO

CREATE UNIQUE CLUSTERED INDEX NoPgSplit_ind on NoPgSplit(col1)
GO

--Insert Rows in Clustered-Index-Order.
SET NOCOUNT ON
GO
DECLARE @i int
SET @i=0
WHILE (@i<10000)
BEGIN
   INSERT INTO NoPgSplit VALUES(@i,'___')
   SET @i=@i+1
END
GO
SET NOCOUNT OFF
GO
```

Question N: What was the max page splits/sec in perfmon?

Question O: What was the difference between this value and the value seen in Step 3?

Question P: Even though we insert the rows in clustered-index-order, we still see page splits in perfmon. What does this tell you about this counter?

Question Q: Will you see a page split only when you insert data?

Compare the fragmentation between PgSplit and NoPgSplit

15. Execute the SQL script in *6.ExamineFragementation.sql*.

```
use AdventureworksPTO
go
DECLARE @db_id SMALLINT;
DECLARE @object_id INT;

SET @db_id = DB_ID(N'AdventureWorksPTO');
SET @object_id = OBJECT_ID(N'AdventureWorksPTO.dbo.NoPgSplit');

SELECT *
FROM sys.dm_db_index_physical_stats(@db_id,
    @object_id, NULL, NULL , 'LIMITED');
go
```

16. Again note the values for avg_fragmentation_in_percent and page_count columns.

Compare the IO differences between simple queries on both tables

17. Execute the SQL script in *7.CompareStatisticsIO.sql*.

```
Use AdventureworksPTO
go

SET STATISTICS IO ON
GO
```

```
PRINT 'PgSplit Performance:'
PRINT '===================='
SELECT *
FROM PgSplit WHERE col2='a'

PRINT 'NoPgSplit Performance:'
PRINT '======================'
SELECT *
FROM NoPgSplit
WHERE col2='a'
GO

SET STATISTICS IO OFF
GO
```

Question R: How many logical reads are needed for the same query as above on the NoPgSplit table?

Question S: Why would there be fewer logical reads? (Comparing the output from dm_db_index_physical_stats on both tables should provide the answer.) Rerun ExamineFragmentation.sql if you need help.

Use Alter Index to remove fragmentation

18. Run *8.AlterIdx_PgSplit.sql*.

```
use AdventureWorksPTO
go

alter index PgSplit_ind on PgSplit Rebuild
GO
```

19. Execute the SQL script in *9.ExamineFragmentation.sql*.

```
use AdventureworksPTO
go
DECLARE @db_id SMALLINT;
DECLARE @object_id INT;
```

```
SET @db_id = DB_ID(N'AdventureWorksPTO');
SET @object_id = OBJECT_ID(N'AdventureWorksPTO.dbo.PgSplit');

SELECT *
FROM sys.dm_db_index_physical_stats(@db_id,
   @object_id, NULL, NULL , 'LIMITED');
go
```

Question T: What are the values for avg_fragmentation_in_percent and page_count?

20. Execute the SQL script in *10.ShowStatisticsIO.sql* and compare the logical reads vs. the number of logical reads for PgSplit in step 12.

```
Use AdventureworksPTO
go

SET STATISTICS IO ON
PRINT 'PgSplit Performance:'
PRINT '===================='

SELECT *
FROM PgSplit
WHERE col2='a'

SET STATISTICS IO OFF
```