Clustered Index Internals

vLabs
by SQLMaestros

Lead Content Developer:

Amit R S Bansal (@A_Bansal | www.amitbansal.net)
Microsoft Certified Master of SQL Server
Microsoft Most Valuable Professional for SQL Server

Table of Contents

Before You Begin

Estimated time to complete this lab

40 minutes

Objectives:

After completing this lab, you will be able to:

- Understand the Clustered Index internals in SQL Server
- Understand Clustered index architecture
- Understand Page split and fragmentation
- Understand Clustered index key
- Understand Ordering of data in clustered index

Prerequisites

Before working on this lab, you must have:

- Basic administration experience with SQL Server

Lab scenario

In SQL Server, indexes are organized as B-trees. Each page in an index B-tree is called an index node. The top node of the B-tree is called the root node. The bottom level of nodes in an index is called as leaf nodes. Any index levels between the root and the leaf nodes are collectively known as intermediate levels. In a clustered index, the leaf nodes contain the data pages of the underlying table. The root and intermediate level nodes contain index pages holding index rows. Each index row contains a key value and a pointer to either an intermediate level page in the B-tree, or a data row in the leaf level of the index. The pages in each level of the index are linked in a doubly-linked list. In the first exercise we will observe B-Tree structure of a clustered index. In the second exercise we will understand the concept of page split and how page split causes fragmentation in case of clustered index. In exercise we will explore the significance of choosing proper clustered index key and in the fourth exercise we will look at the ordering of data in case of clustered index.

Tips to complete this lab successfully

Following these tips will be helpful in completing the lab successfully in time

- All lab files are located in C:\vLabs\ Clustered_Index_Internals folder
- The script(s) are divided into various sections marked with 'Begin', 'End' and 'Steps'. As per the instructions, execute the statements between particular sections only or for a particular step
- Read the instructions carefully and do not deviate from the flow of the lab
- In case you execute the entire script by mistake or miss a step or get confused midway, simply 'Restart' the VM from the VM control panel to restart/redo the lab

Exercise 1: Understanding Clustered Index B-Tree Structure

Scenario

In this exercise, we will look at clustered index internals, how B-Tree is formed and what kind of data resides in each level of the clustered index.

Tasks	Detailed Steps
Launch **SQL Server Management Studio**	1. Click **Start** | **All Programs** | **SQL Server 2012** | **SQL Server Management Studio** or Double click **SQL Server Management Studio** shortcut on the desktop 2. In the **Connect to Server** dialog box, click **Connect**
Open **1_UnderstandingClusteredIndexInternals.sql**	1. Click **File** | **Open** | **File** or press **(Ctrl + O)** 2. Navigate to C:\vLabs\ Clustered_Index_Internals 3. Select **1_UnderstandingClusteredIndexInternals.sql** and click **Open**
Execute the statement(s) in the 'Setup' section to setup the database and table	The setup section performs the following: • **SQLMaestros** database is created • **SQLMaestros** schema is created • **Table1** table is created with 1000 records In **1_UnderstandingClusteredIndexInternals.sql**, Review and execute the statement(s) in section 'Begin: Setup' and 'End: Setup'

```
---------------------
-- Begin: Setup
---------------------

-- Create a database named SQLMaestros
USE master;
GO
IF EXISTS(SELECT 1 FROM sys.databases WHERE name='SQLMaestros')
ALTER DATABASE [SQLMaestros] SET  SINGLE_USER WITH ROLLBACK IMMEDIATE;
DROP DATABASE SQLMaestros;
CREATE DATABASE SQLMaestros;
GO


USE SQLMaestros;
SET NOCOUNT ON;
GO

-- Create a schema named SQLMaestros
CREATE SCHEMA [SQLMaestros] AUTHORIZATION [dbo];
GO

-- Create Table1 Table in SQLMaestros database
CREATE Table [SQLMaestros].[Table1](
   Column1 INT,
   Column2 VARCHAR(8000),
   Column3 CHAR(10),
   Column4 INT);
GO

-- Insert 1000 records into Table1 table
DECLARE @COUNT INT;
SET @COUNT = 1;
DECLARE @DATA1 VARCHAR(7000)
SET @DATA1 = REPLICATE('bigdata',1000)
WHILE @COUNT < 1001
BEGIN
DECLARE @DATA2 INT;
SET @DATA2 = ROUND(10000000*RAND(),0);
```

```
INSERT INTO [SQLMaestros].[Table1] VALUES(@COUNT,@DATA1,'AAAAA',@DATA2);
SET @COUNT = @COUNT + 1;
END
GO

---------------------
-- End: Setup
---------------------
```

CREATE a clustered index

Execute the following statement(s) to **CREATE** a clustered index on **Column1** column of **Table1** table

```
-- Step 1: Create a clustered index on Column1 column of Table1 table
CREATE CLUSTERED INDEX CL_Table1_Column1 ON [SQLMaestros].[Table1](Column1);
GO
```

Note: There are many clauses that you can specify during index creation. Many of these clauses are optional. If we don't specify, SQL server will accept the default values. In case, we want to use these clauses, we have to use the syntax **WITH** (**OPTION_NAME = VALUE**).

```
  | PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor (Integer value between 0 – 100)
  | SORT_IN_TEMPDB = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | DROP_EXISTING = { ON | OFF }
  | ONLINE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism (Integer value depending upon the no. of CPU)
  | DATA_COMPRESSION = { NONE | ROW | PAGE}
```

View index details	Execute the following statement(s) to view index details of **Table1** table `-- Step 2: View index information for Table1 Table` `EXEC sp_helpindex 'SQLMaestros.Table1';` `GO` **Note: sp_helpindex** system stored procedure can be used to view index details for a particular table.
View statistics details	Execute the following statement(s) to view the statistics details of **Table1** table `-- Step 3: View statistics information for Table Table1` `SELECT STATS.* FROM sys.stats AS STATS` `INNER JOIN sys.objects AS OBJ` `ON STATS.object_id = OBJ.object_id` `WHERE OBJ.name = 'Table1';` `GO` Editor Results Messages |	object_id	name	stats_id	auto_created	user_created	no_recompute	has_filter | |---|---|---|---|---|---|---|---| | 1 | 885578193 | CL_Table1_Column1 | 1 | 0 | 0 | 0 | 0 | **Note:** Whenever we create an index, SQL Server automatically creates statistics for that index. If a statistics object is manually created then **user_created** column will be **'1**'.Apart from statistics being created automatically for an index, SQL Server can automatically create statistics for a column without an index if that column is used in a predicate and **AUTO_CREATE_STATISTICS** database option is 'ON'. If that is the case, **auto_created** column will show a value of '**1'**.

View clustered index details

Execute the following statement(s) to view clustered index details of **Table1** table

```
-- Step 4: View clustered index details
SELECT
index_id,index_type_desc,index_level,page_count,avg_record_size_in_bytes,avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats
    (DB_ID(N'SQLMaestros'), OBJECT_ID(N'SQLMaestros.Table1'), 1, NULL , 'DETAILED')
ORDER BY index_level DESC;
GO
```

Results | Messages

	index_id	index_type_desc	index_level	page_count	avg_record_size_in_bytes	avg_fragmentation_in_percent
1	1	CLUSTERED INDEX	2	1	14	0
2	1	CLUSTERED INDEX	1	2	14	0
3	1	CLUSTERED INDEX	0	1000	7031	0

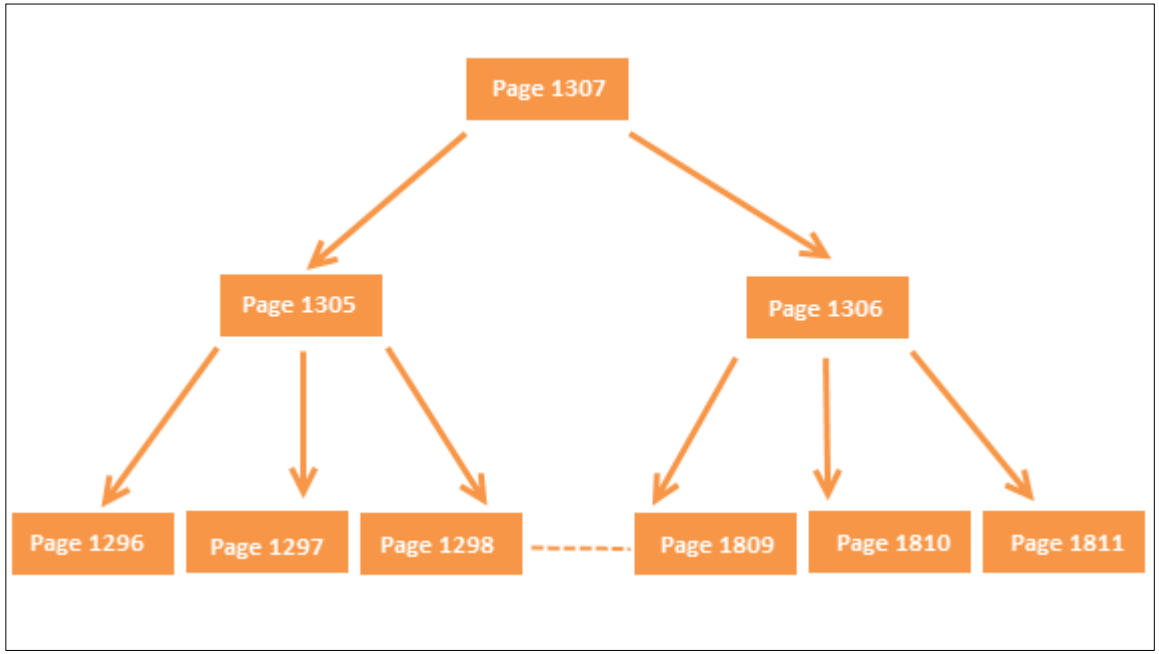
Note: We can use **sys.dm_db_index_physical_stats()** DMF to get detailed index information. Below is the complete list of parameters that we can pass to this DMF:

```
sys.dm_db_index_physical_stats
      (
          { database_id | NULL | 0 | DEFAULT }
        , { object_id | NULL | 0 | DEFAULT }
        , { index_id | NULL | 0 | -1 | DEFAULT }
        , { partition_number | NULL | 0 | DEFAULT }
        , { mode [DETAILED|SAMPLED|LIMITED] | NULL | DEFAULT }
      )
```

Observation: We are using **sys.dm_db_index_physical_stats** Dynamic Management Object to view index metadata. In the above output, **index_level** column represents the index depth. **Index_level** '**0**' is for leaf level and any subsequent higher value represents the intermediate level and root level. The clustered index has three levels. First row is for the root level (**index_level = 2**), second row for intermediate level (**index_level = 1**) and third row for leaf level (**index_level = 0**). **page_count**,

	avg_record_size_in_bytes and **avg_fragmentation_in_percent** represent no. of pages, average row size in each page and amount of fragmentation in each level respectively. **Note: Fillfactor** is only applicable for leaf level pages. If you want to define index fillfactor to intermediate and root level, then we have to specify that by turning **Pad_Index** option 'ON' while creating or rebuilding the index.
View B-Tree structure of the clustered index	Execute the following statement(s) to view the B-Tree structure of the clustered index of **Table1** table `-- Step 5: View clustered index architecture` `SELECT allocated_page_page_id,page_type_desc,page_level,next_page_page_id,previous_page_page_id` `FROM sys.dm_db_database_page_allocations(DB_ID(N'SQLMaestros'), OBJECT_ID(N'SQLMaestros.Table1'), NULL, NULL, 'DETAILED')` `WHERE page_type IN (1,2)` `ORDER BY page_level DESC;` `GO` Results Messages

	allocated_page_page_id	page_type_desc	page_level	next_page_page_id	previous_page_page_id
1	1307	INDEX_PAGE	2	NULL	NULL
2	1305	INDEX_PAGE	1	1306	NULL
3	1306	INDEX_PAGE	1	NULL	1305
4	1296	DATA_PAGE	0	1297	NULL
5	1297	DATA_PAGE	0	1298	1296
6	1298	DATA_PAGE	0	1299	1297
7	1299	DATA_PAGE	0	1300	1298



Note: sys.dm_db_database_page_allocations() is an undocumented DMF available only in SQL Server 2012. Below is the parameter list that can be passed into this DMF

```
sys.dm_db_database_page_allocations
      (
          { database_id | NULL | DB_ID() }
        , { object_id | NULL | OBJECT_ID() }
        , { index_id | NULL }
        , { partition_number | NULL }
        , { mode [DETAILED|LIMITED] | NULL | DEFAULT }
       )
```

View memory dump of root level page

Execute the following statement(s) to view memory dump of clustered index root page (Replace 1307 in the below statement with **allocated_page_page_id** of the root page from the output of **step 5**[**Note**: For root page **page_level** is **2**])

```
-- Step 6: View memory dump of root page
DBCC TRACEON(3604);
DBCC PAGE('SQLMaestros',1,1307,3); -- Page ID will change in your case
GO
```

Results | Messages

	FileId	PageId	Row	Level	ChildFileId	ChildPageId	COL1 (key)	UNIQUIFIER (key)	KeyHashValue	Row Size
1	1	1307	0	2	1	1305	NULL	NULL	NULL	14
2	1	1307	1	2	1	1306	506	0	NULL	14

Note: Clustered index root page dose not contains any user data but only pointers to the intermediate level pages. We can observer that in the Messages section in the output.

Note: In order to view DBCC PAGE output in SSMS we have to enable Trace Flag 3604. DBCC PAGE() command can be used to view a page contents. Below is the complete parameter list that we can pass into DBCC PAGE() command:

```
DBCC  PAGE
       (
          { database_name | database_id | DB_ID() }
        , { file_number }
        , { page_number }
         , { print_option [0 – header | 1 – header + slot array | 2 – header + whole page hex dump | 3 -
        header + complete page hex dump with row by row interpretation}
        )
```

View memory dump of intermediate level page

Execute the following statement(s) to view memory dump of a clustered index intermediate level page (Replace 1305 in the below statement with **allocated_page_page_id** of a particular intermediate level page from the output of **step 5**[**Note**: For Intermediate level page **page_level** is **1**])

```
--Step 7: View memory dump of intermediate level page
DBCC PAGE('SQLMaestros',1,1305,3); -- Page ID will change in your case
GO
```

Editor | Results | Messages

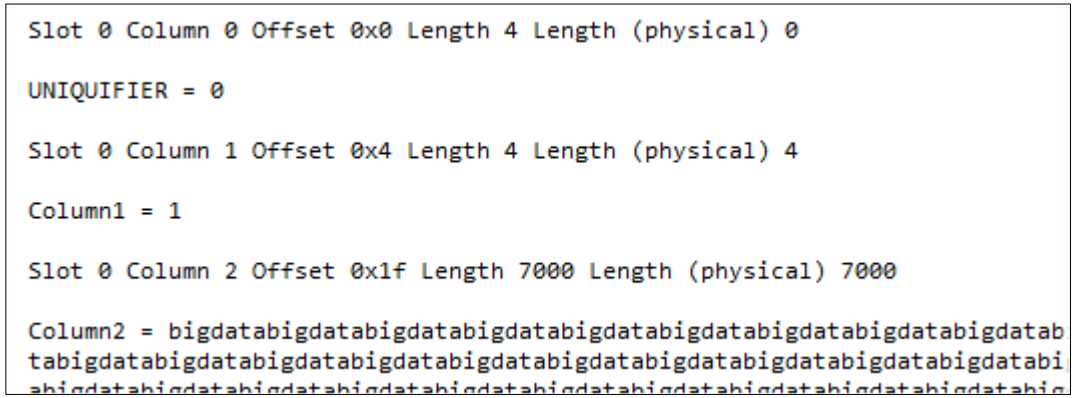
	FileId	PageId	Row	Level	ChildFileId	ChildPageId	Column1 (key)	UNIQUIFIER (key)	KeyHashValue	Row Size
1	1	1305	0	1	1	1296	NULL	NULL	NULL	14
2	1	1305	1	1	1	1297	2	0	NULL	14
3	1	1305	2	1	1	1298	3	0	NULL	14
4	1	1305	3	1	1	1299	4	0	NULL	14
5	1	1305	4	1	1	1300	5	0	NULL	14
6	1	1305	5	1	1	1301	6	0	NULL	14
7	1	1305	6	1	1	1302	7	0	NULL	14

Observation: As in case of root page, intermediate level pages in a clustered index also does not contains any data but pointers to the leaf level pages and clustered index key. We have created a non-unique clustered index, thus to make all the clustered index keys unique SQL Server has added an **UNIQUIFIER (key).**

View memory dump of leaf level page

Execute the following statement(s) to view memory dump of clustered index leaf level page (Replace 1296 in the below statement with **allocated_page_page_id** of a particular leaf level page from the output of **step 5**[Note: For leaf level page **page_level** is **0**])

```
--Step 8: View memory dump of leaf level page
DBCC PAGE('SQLMaestros',1,1296,3); -- Page ID will change in your case
GO
```

	 **Observation:** Leaf level pages of a clustered index contain the actual table data itself.
Close all the query windows	Close all the query windows () and if **SSMS** asks to save changes, click **NO**

Summary

In this exercise, you have learnt:

- Different clauses that you can specify during the creation\rebuilding of a clustered index
- How to view clustered index B-Tree structure using DMVs & DMFs
- Different levels of clustered index b-tree structure and what they contain
- About Index statistics
- View index information from sys.dm_db_index_physical_stats() DMF

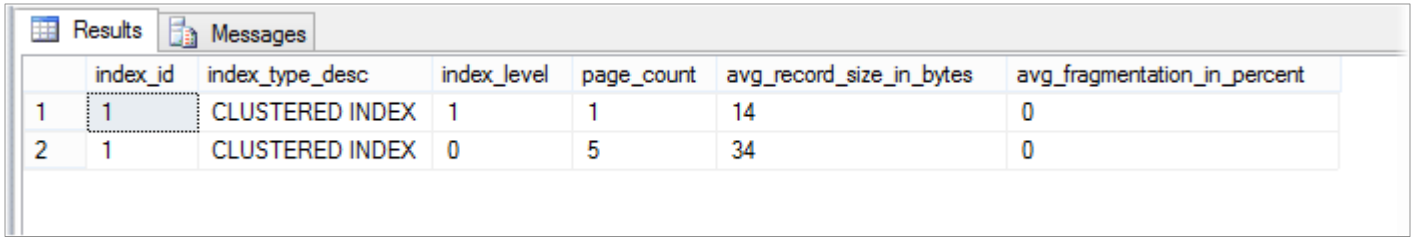
Exercise 2: Page Splits and Fragmentation in Clustered Index

Scenario

In this exercise, we will look at page split in clustered index and how they generate fragmentation.

Tasks	Detailed Steps
Open **2_PageSplitFragmentation.sql**	1. Click **File** | **Open** | **File** or press (Ctrl + O) 2. Navigate to C:\vLabs\ Clustered_Index_Internals 3. Select **2_PageSplitFragmentation.sql** and click **Open**
Execute the statement(s) in the 'Setup' section to create a table	The setup section performs the following: • **Table2** table is created with 1000 records In **2_PageSplitFragmentation.sql**, Review and execute the statement(s) in section 'Begin: Setup' and 'End: Setup' `----------` `-- Begin: Setup` `----------` `USE SQLMaestros;` `SET NOCOUNT ON;` `GO` `-- Create Table2 table in SQLMaestros database` `CREATE Table [SQLMaestros].[Table2](` `  Column1 INT,` `  Column2 VARCHAR(8000),` `  Column3 DATETIME);` `GO`

	`` -- Insert 1000 records in Table2 table DECLARE @COUNT INT; SET @COUNT = 1; WHILE @COUNT < 1001 BEGIN INSERT INTO [SQLMaestros].[Table2] VALUES(@COUNT,'smalldata',GETDATE()); SET @COUNT = @COUNT + 1; END GO --------------------- -- End: Setup --------------------- ```
CREATE a clustered index	Execute the following statement(s) to **CREATE** a clustered index on **Column1** column of **Table1** table ``` -- Step 1: Create clustered index on Column1 column of Table2 table CREATE CLUSTERED INDEX CL_Table1_Column1 ON [SQLMaestros].[Table2](Column1); GO ```
View clustered index details	Execute the following statement(s) to view clustered index details of **Table2** table ``` -- Step 2: View clustered index details SELECT index_id,index_type_desc,index_level,page_count,avg_record_size_in_bytes,avg_fragmentation_in_percent FROM sys.dm_db_index_physical_stats (DB_ID(N'SQLMaestros'), OBJECT_ID(N'SQLMaestros.Table2'), 1, NULL , 'DETAILED') ORDER BY index_level DESC; GO ```



Results | Messages

	index_id	index_type_desc	index_level	page_count	avg_record_size_in_bytes	avg_fragmentation_in_percent
1	1	CLUSTERED INDEX	1	1	14	0
2	1	CLUSTERED INDEX	0	5	34	0

Observation: Currently the clustered index has 0% fragmentation and there are five leaf level (data) pages and the root page.

View page split details

Execute the following statement(s) to view the no. of page splits in **SQLMaestros** database using **fn_dblog()** function

```
-- Step 3: View page split details from log file
Select COUNT(1) AS NumberOfSplits, AllocUnitName , Context
From fn_dblog(NULL,NULL)
Where operation = 'LOP_DELETE_SPLIT'
Group By AllocUnitName, Context
Order by NumberOfSplits desc
```

Note: **fn_dblog()** is an undocumented function that we can use to view transactional log file content. Below is the complete parameter list that we can pass to this function:

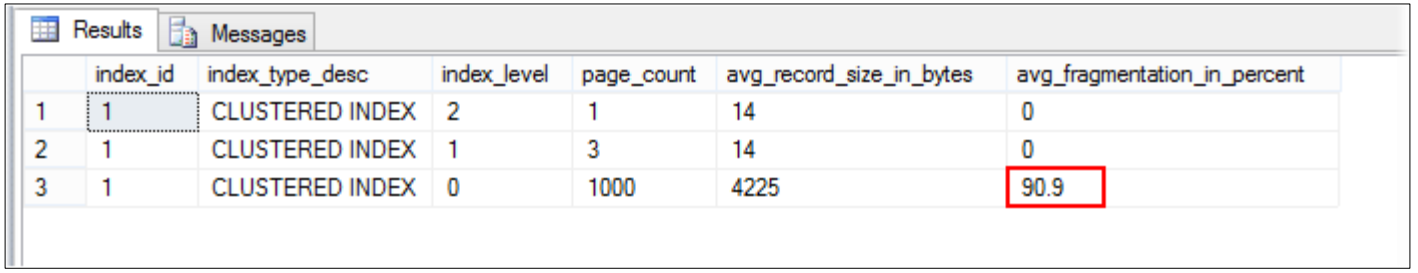
```
fn_dblog
        (
        { start_time | NULL }
      ,{ end_time | NULL }
        )
```

UPDATE records in Table2

Execute the following statement(s) to **UPDATE Column2** column of **Table2** table

```
-- Step 4: Perform update operation to split the pages
DECLARE @DATA1 VARCHAR(4200)
SET @DATA1 = REPLICATE('bigdata',600)
UPDATE [SQLMaestros].[Table2] SET Column2 = @DATA1
```

	`` WHERE Column1 % 2 = 0 DECLARE @DATA2 VARCHAR(4200) SET @DATA2 = REPLICATE('bigdata',600) UPDATE [SQLMaestros].[Table2] SET Column2 = @DATA2 WHERE Column1 % 2 = 1 ``` **Explanation:** A data page can contain maximum 8060 bytes of data. In the above workload, we are updating **Column2** to 4200 bytes, which means all the records need to move to another page in order to successfully **UPDATE** a single row of **Column2** value. Essentially, each page now can only contain a single record. Five leaf level pages will be split into 1000 leaf level pages and subsequent intermediate level page will be created due to a page split in the root page as well.
View page split details	Execute the following statement(s) to view the no. of page splits in **SQLMaestros** database using **fn_dblog()** function ``` -- Step 5: View page split details from log file Select COUNT(1) AS NumberOfSplits, AllocUnitName , Context From fn_dblog(NULL,NULL) Where operation = 'LOP_DELETE_SPLIT' Group By AllocUnitName, Context Order by NumberOfSplits desc ``` **Note:** Note the number of page splits that occurred
View clustered index details	Execute the following statement(s) to view clustered index details for **Table2** table ``` -- Step 6: View clustered index details SELECT index_id,index_type_desc,index_level,page_count,avg_record_size_in_bytes,avg_fragmentation_in_percent FROM sys.dm_db_index_physical_stats (DB_ID(N'SQLMaestros'), OBJECT_ID(N'SQLMaestros.Table2'), 1, NULL , 'DETAILED') ORDER BY index_level DESC; GO ```

	 **Observation:** Before the **UPDATE** operation the clustered index had two levels with one root page and five leaf level pages with 0% fragmentation. Due to the **UPDATE** operation leaf level pages split and 995 new pages are created. Before page split, root page had to hold five records (pointers to 5 pages below it) but due to page splits it has to now hold 1000 records (one for each for leaf level page). But the average row size in root page is 14 bytes. So in order to accommodate all pointers for leaf level pages, the root page had to hold 14 X 1000 = 14000 bytes of data. As we already know that a page can contain maximum 8060 bytes of data root page had to split and intermediate level pages are created to accommodate the **UPDATE** operation. Also notice that due to page splits, leaf level page fragmentation increased from 0 to 90 percent, approximately. **Tips**: In order to fix fragmentation, the index can be re-built or de-frag(ed).
Close all the query windows	Close all the query windows ( ) and if **SSMS** asks to save changes, click **NO**

	index_id	index_type_desc	index_level	page_count	avg_record_size_in_bytes	avg_fragmentation_in_percent
1	1	CLUSTERED INDEX	2	1	14	0
2	1	CLUSTERED INDEX	1	3	14	0
3	1	CLUSTERED INDEX	0	1000	4225	90.9

Summary

In this exercise, you have learnt:

- The concept of page splits and how/when they can occur
- How fragmentation increases due to page splits
- How to use fn_dblog() function to monitor page splits
- How to use sys.dm_db_database_page_allocations DMF to view pages allocated to a table
- How to use sys.dm_db_index_physical_stats DMF to view index metadata

Exercise 3: Clustered Index Key

Scenario

In this exercise, we will look at clustered index key and how important it is to select the perfect key for clustered index.

Tasks	Detailed Steps
Open **3_ClusteredIndexKey.sql**	1. Click **File** | **Open** | **File** or press (Ctrl + O) 2. Navigate to C:\vLabs\ Clustered_Index_Internals 3. Select **3_ClusteredIndexKey.sql** and click **Open**
Execute the statement(s) in the 'Setup' section to setup the table Table3	The setup section performs the following: • **Table3** table is created with 1000 records • **Column6** column of **Table3** table is updated to contain some **duplicate** values**.** **Note:** We are purposely inserting **duplicate** values to demonstrate the concept of **UNIQUIFIER(key)** In **3_ClusteredIndexKey.sql**, review and execute the statements in section 'Begin: Setup' and 'End: Setup' `----------` `-- Begin: Setup` `----------` `USE SQLMaestros;` `SET NOCOUNT ON;` `GO` `-- Create Table3 table in SQLMaestros database` `CREATE Table [SQLMaestros].[Table3](` `Column1 INT,`

```
	Column2 CHAR(15),
	Column3 VARCHAR(100),
	Column4 UNIQUEIDENTIFIER,
	Column5 DATETIME,
	Column6 INT);

-- Insert 1000 records in Table3 table
DECLARE @COUNT INT;
SET @COUNT = 1;
DECLARE @DATA1 VARCHAR(100)
SET @DATA1 = REPLICATE('data',25)
WHILE @COUNT < 1001
BEGIN
DECLARE @DATA2 INT;
SET @DATA2 = ROUND(10000000*RAND(),0);
INSERT INTO [SQLMaestros].[Table3] VALUES(@COUNT, 'SQLMaestros', @DATA1, NEWID(), GETDATE(),
@DATA2);
SET @COUNT = @COUNT + 1;
END
GO

-- Update Column6 column of Table3 table
UPDATE [SQLMaestros].[Table3] SET Column6 = 100
WHERE Column1 % 5 = 0

----------------------
-- End: Setup
----------------------
```

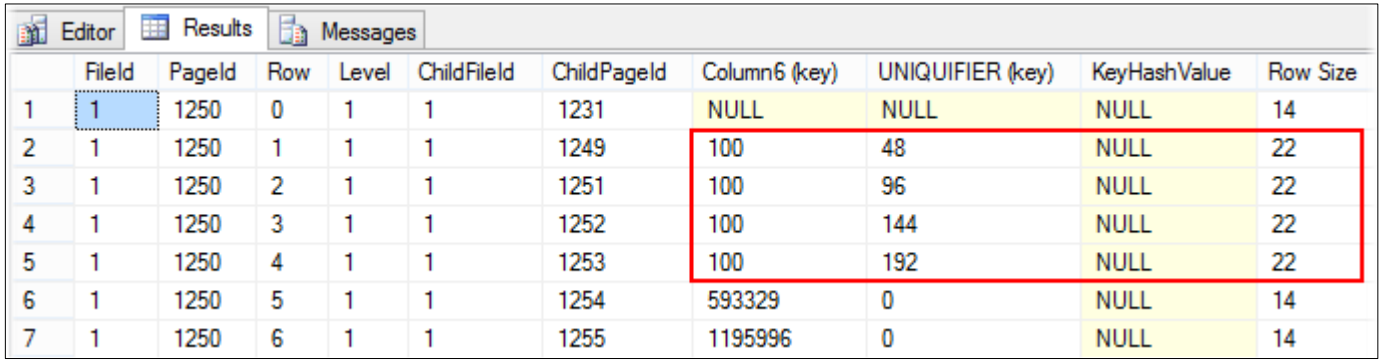
CREATE a clustered index

Execute the following statement(s) to **CREATE** a clustered index on **Column6** column of **Table3** table

```
-- Step 1: Create a clustered index on Column6 column of Table3 table
CREATE CLUSTERED INDEX CL_Table3_Column6 ON [SQLMaestros].[Table3](Column6);
GO
```

Note: Column6 column of **Table3** table contains duplicate values.

View pages allocations	Execute the following statement(s) to view pages allocated to the clustered index on **Table3** table `-- Step 2: View pages allocated to clustered index in Table3 table` `SELECT allocated_page_page_id,page_type_desc,page_level,next_page_page_id,previous_page_page_id` `FROM sys.dm_db_database_page_allocations(DB_ID(N'SQLMaestros'), OBJECT_ID(N'SQLMaestros.Table3'),` `1, NULL, 'DETAILED')` `WHERE page_type IN (1,2)` `ORDER BY page_level DESC;` `GO` Results Messages allocated_page_page_id / page_type_desc / page_level / next_page_page_id / previous_page_page_id 1 1250 INDEX_PAGE 1 NULL NULL 2 1251 DATA_PAGE 0 1252 1249 3 1252 DATA_PAGE 0 1253 1251 4 1253 DATA_PAGE 0 1254 1252 5 1254 DATA_PAGE 0 1255 1253 6 1255 DATA_PAGE 0 1256 1254 7 1256 DATA_PAGE 0 1257 1255 8 1257 DATA_PAGE 0 1258 1256
View memory dumps of root page	Execute the following statement(s) to view memory dump of the root page of the clustered index (Replace 1250 in the below statement with **allocated_page_page_id** of the root level page from the output of **step 2**[**Note**: For root level page **page_level** is **1**]) `-- Step 3: View memory dump of the root page` `DBCC TRACEON(3604);` `GO` `DBCC PAGE('SQLMaestros',1,1250,3); -- Page ID will change in your case` `GO`



	FileId	PageId	Row	Level	ChildFileId	ChildPageId	Column6 (key)	UNIQUIFIER (key)	KeyHashValue	Row Size
1	1	1250	0	1	1	1231	NULL	NULL	NULL	14
2	1	1250	1	1	1	1249	100	48	NULL	22
3	1	1250	2	1	1	1251	100	96	NULL	22
4	1	1250	3	1	1	1252	100	144	NULL	22
5	1	1250	4	1	1	1253	100	192	NULL	22
6	1	1250	5	1	1	1254	593329	0	NULL	14
7	1	1250	6	1	1	1255	1195996	0	NULL	14

Explanation: 20 % of **Column6** column data was updated to 100 in the setup section. We then created a clustered index on **Column6** column. In a clustered index, SQL Server database engine needs to identify each value uniquely so that lookups from non-clustered indexes are performed correctly. This unique value is the clustering key. But **Column6** column contains duplicate values so in order to uniquely identify each value; SQL Server inserts an **UNIQUIFIER (key).** **UNIQUIFIER (key)** is only added for the first duplicate value. Because to the SQL Server the first 100 is unique and then onwards whenever it finds a 100 it will add an **UNIQUIFIER (key)** and will increment that value. Thus in the above output 48 in the **UNIQUIFIER (key)** column represents 49th 100, 96 in the **UNIQUIFIER (key)** column represents 97th 100 and so on.

View memory dump of a leaf level page

Execute the following statement(s) to view the leaf level page of the clustered index (Replace 1251 in the below statement with **allocated_page_page_id** of a leaf level page from the output of **step 2**[**Note**: For leaf level page **page_level** is **0**])

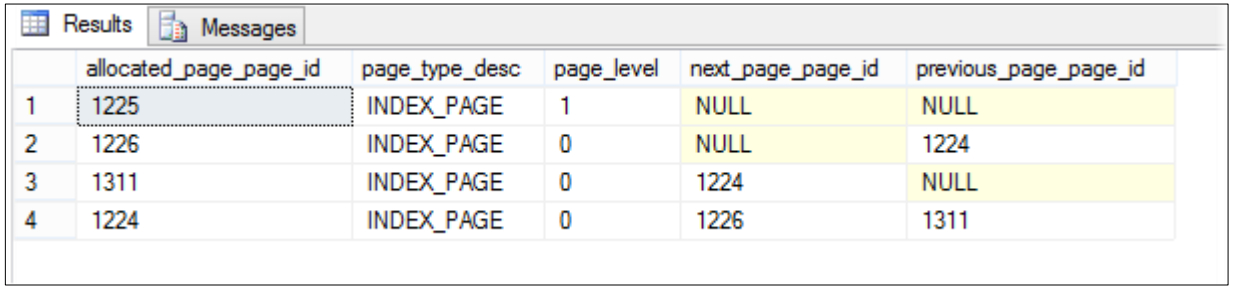
```
-- Step 4: View memory dump of a particular leaf page (data page)
DBCC PAGE('SQLMaestros',1,1251,3); -- Page ID will change in your case
GO
```

```
Slot 0 Offset 0x60 Length 164

Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS
Record Size = 164
Memory Dump @0x000000006F95A060

0000000000000000:   30003300 64000000 e5010000 53514c4d 61657374  0.3.d...å...SQLMaest
0000000000000014:   726f7320 20202046 bf356c48 5a764888 937c5446  ros    F¿5lHZvH.|TF
0000000000000028:   abffc0e0 32e300e4 a2000007 00000200 4000a400  «ÿÀà2ã.ä¢.......@.¤.
000000000000003C:   60000000 64617461 64617461 64617461 64617461  `...datadatadatadata
0000000000000050:   64617461 64617461 64617461 64617461 64617461  datadatadatadatadata
0000000000000064:   64617461 64617461 64617461 64617461 64617461  datadatadatadatadata
0000000000000078:   64617461 64617461 64617461 64617461 64617461  datadatadatadatadata
000000000000008C:   64617461 64617461 64617461 64617461 64617461  datadatadatadatadata
00000000000000A0:   64617461                                      data

Slot 0 Column 0 Offset 0x3c Length 4 Length (physical) 4

UNIQUIFIER = 96

Slot 0 Column 6 Offset 0x4 Length 4 Length (physical) 4

Column6 = 100

Slot 0 Column 1 Offset 0x8 Length 4 Length (physical) 4
```

Explanation: Scroll down and locate the **UNIQUIFIER**

Note: If we create clustered index on a column with duplicate value, there will be 4 bytes extra required per row to include the **UNIQUIFIER (key).**

CREATE a non-clustered index	Execute the following statement(s) to **CREATE** a non-clustered index on **Column5** column of **Table3** table `-- Step 5: Create a non-clustered index on Column5 column of Table3 table` `CREATE NONCLUSTERED INDEX NCL_Table3_Column5 ON [SQLMaestros].[Table3](Column5);` `GO`
View index id of non-clustered index	Execute the following statement(s) to view index id of the non-clustered index `-- Step 6: Find the index id of non-clustered index` `SELECT name, index_id FROM sys.indexes` `WHERE name = 'NCL_Table3_Column5';` `GO` **Note:** In case of a clustered index, **index_id** will always be '1' and non-clustered indexes will have their index_ids starting from 2 onwards
View page allocation	Execute the following statement(s) to view the pages allocated to the non-clustered index on **Table3** table(Replace **<index_id>** in the below statement with **index_id** we get in **step 6**) `-- Step 7: View pages allocated to non-clustered index on Table3 table` `SELECT allocated_page_page_id,page_type_desc,page_level,next_page_page_id,previous_page_page_id` `FROM sys.dm_db_database_page_allocations(DB_ID(N'SQLMaestros'), OBJECT_ID(N'SQLMaestros.Table3'),` `<index_id>, NULL, 'DETAILED')` `WHERE page_type IN (1,2)` `ORDER BY page_level DESC;` `GO`



Results | Messages

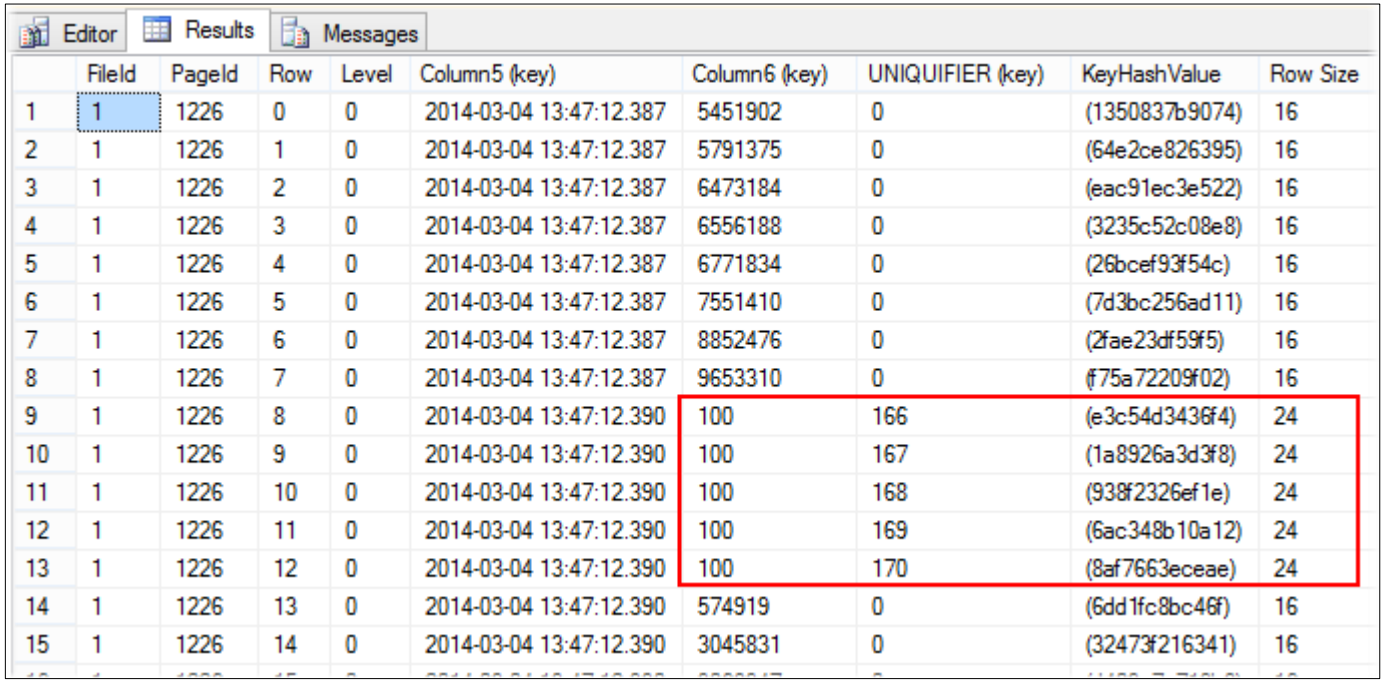
	allocated_page_page_id	page_type_desc	page_level	next_page_page_id	previous_page_page_id
1	1225	INDEX_PAGE	1	NULL	NULL
2	1226	INDEX_PAGE	0	NULL	1224
3	1311	INDEX_PAGE	0	1224	NULL
4	1224	INDEX_PAGE	0	1226	1311

Observation: Four pages are allocated to the non-clustered index (One root page and three leaf page).

View memory dump leaf level page

Execute the following statement(s) to view non-clustered index leaf level page memory dump (Replace 1226 in the below statement with **allocated_page_page_id** of a leaf level page from the output of **step 7**[**Note**: For leaf level page **page_level** is **0**])

```
-- Step 8: View non-clustered index leaf level page
DBCC PAGE('SQLMaestros',1,1226,3);    -- Page ID will change in your case
GO
```



	FileId	PageId	Row	Level	Column5 (key)	Column6 (key)	UNIQUIFIER (key)	KeyHashValue	Row Size
1	1	1226	0	0	2014-03-04 13:47:12.387	5451902	0	(1350837b9074)	16
2	1	1226	1	0	2014-03-04 13:47:12.387	5791375	0	(64e2ce826395)	16
3	1	1226	2	0	2014-03-04 13:47:12.387	6473184	0	(eac91ec3e522)	16
4	1	1226	3	0	2014-03-04 13:47:12.387	6556188	0	(3235c52c08e8)	16
5	1	1226	4	0	2014-03-04 13:47:12.387	6771834	0	(26bcef93f54c)	16
6	1	1226	5	0	2014-03-04 13:47:12.387	7551410	0	(7d3bc256ad11)	16
7	1	1226	6	0	2014-03-04 13:47:12.387	8852476	0	(2fae23df59f5)	16
8	1	1226	7	0	2014-03-04 13:47:12.387	9653310	0	(f75a72209f02)	16
9	1	1226	8	0	2014-03-04 13:47:12.390	100	166	(e3c54d3436f4)	24
10	1	1226	9	0	2014-03-04 13:47:12.390	100	167	(1a8926a3d3f8)	24
11	1	1226	10	0	2014-03-04 13:47:12.390	100	168	(938f2326ef1e)	24
12	1	1226	11	0	2014-03-04 13:47:12.390	100	169	(6ac348b10a12)	24
13	1	1226	12	0	2014-03-04 13:47:12.390	100	170	(8af7663eceae)	24
14	1	1226	13	0	2014-03-04 13:47:12.390	574919	0	(6dd1fc8bc46f)	16
15	1	1226	14	0	2014-03-04 13:47:12.390	3045831	0	(32473f216341)	16

Explanation: Every non-clustered index on a table will include the row id (in case of heap) or clustered index key in case there is one. Since our clustered index contains **UNIQUIFIER (key),** it will also be included in the non-clustered index. Assuming you have multiple non-clustered indexes on a single table, all of them will include **UNIQUIFIER (key)** and will require 4 bytes extra storage per row.

DROP clustered index

Execute the following statement(s) to **DROP** the clustered index on **Table3** table

```
-- Step 9: Drop Clustered Index on Table3
DROP INDEX CL_Table3_Column6 ON [SQLMaestros].[Table3];
GO
```

CREATE a **UNIQUE** clustered index

Execute the following statement(s) to **CREATE** a **UNIQUE** clustered index on **Column1** column of **Table3** table

```
-- Step 10: Create clustered index on Column1 column of Table3 table
CREATE UNIQUE CLUSTERED INDEX UNCL_Table3_Column1 ON [SQLMaestros].[Table3](Column1);
GO
```

Note: We are creating a **UNIQUE** clustered index so the column can not have any duplicate values.

View pages allocations

Execute the following statement(s) to view pages allocated to the **UNIQUE** clustered index on **Table3** table

```
-- Step 11: View pages allocated to clustered index Table3 table
SELECT allocated_page_page_id,page_type_desc,page_level,next_page_page_id,previous_page_page_id
FROM sys.dm_db_database_page_allocations(DB_ID(N'SQLMaestros'), OBJECT_ID(N'SQLMaestros.Table3'),
1, NULL, 'DETAILED')
WHERE page_type IN (1,2)
ORDER BY page_level DESC;
GO
```

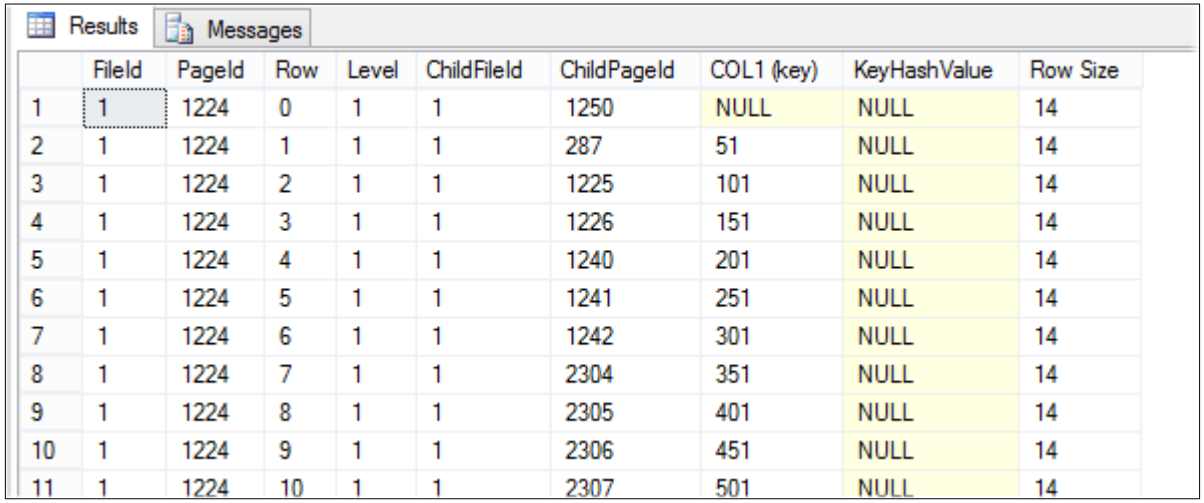
Editor | Results | Messages

	allocated_page_page_id	page_type_desc	page_level	next_page_page_id	previous_page_page_id
1	1224	INDEX_PAGE	1	NULL	NULL
2	1225	DATA_PAGE	0	1226	287
3	1226	DATA_PAGE	0	1240	1225
4	1240	DATA_PAGE	0	1241	1226
5	1241	DATA_PAGE	0	1242	1240

View memory dump of unique clustered index root page

Execute the following statement(s) to view the memory dump of **UNIQUE** clustered index root page. (Replace 1224 in the below statement with **allocated_page_page_id** of the root page from the output of **step 11**[**Note**: For root level page **page_level** is **1**])

```
-- Step 12: View root page
DBCC PAGE('SQLMaestros',1,1224,3);  -- Page ID will change in your case
```



Results | Messages

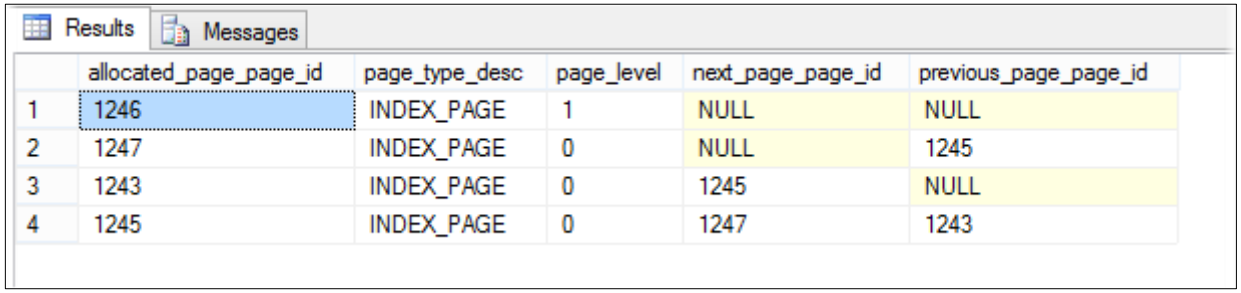
	FileId	PageId	Row	Level	ChildFileId	ChildPageId	COL1 (key)	KeyHashValue	Row Size
1	1	1224	0	1	1	1250	NULL	NULL	14
2	1	1224	1	1	1	287	51	NULL	14
3	1	1224	2	1	1	1225	101	NULL	14
4	1	1224	3	1	1	1226	151	NULL	14
5	1	1224	4	1	1	1240	201	NULL	14
6	1	1224	5	1	1	1241	251	NULL	14
7	1	1224	6	1	1	1242	301	NULL	14
8	1	1224	7	1	1	2304	351	NULL	14
9	1	1224	8	1	1	2305	401	NULL	14
10	1	1224	9	1	1	2306	451	NULL	14
11	1	1224	10	1	1	2307	501	NULL	14

Observation: Since the clustered index is **UNIQUE**, **UNIQUIFIER (key)** is not included this time.

View page allocation

Execute the following statement(s) to view pages allocated to the non-clustered index (Replace the **<index_id>** parameter in the below statement with **index_id** we get from **step 6)**

```
-- Step 13: View pages allocated to non-clustered index on Table3
SELECT allocated_page_page_id,page_type_desc,page_level,next_page_page_id,previous_page_page_id
FROM sys.dm_db_database_page_allocations(DB_ID(N'SQLMaestros'),
OBJECT_ID(N'SQLMaestros.Table3'),<index_id>, NULL, 'DETAILED')
WHERE page_type IN (1,2)
ORDER BY page_level DESC;
```



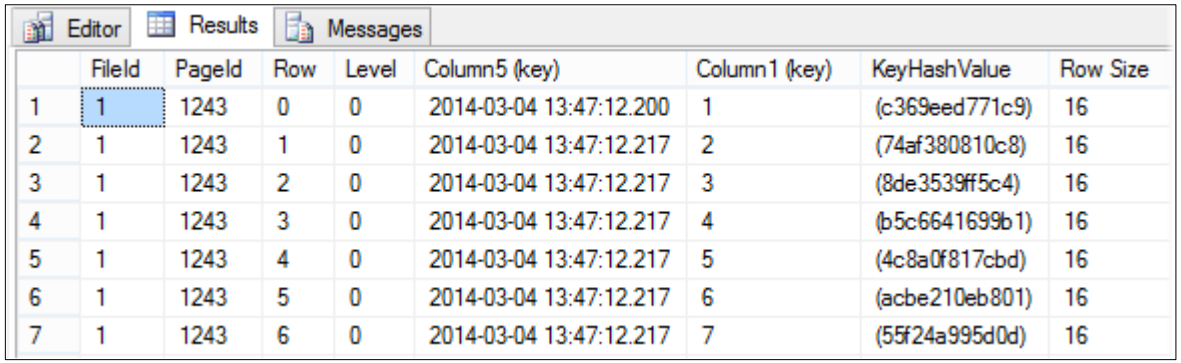
Results | Messages

	allocated_page_page_id	page_type_desc	page_level	next_page_page_id	previous_page_page_id
1	1246	INDEX_PAGE	1	NULL	NULL
2	1247	INDEX_PAGE	0	NULL	1245
3	1243	INDEX_PAGE	0	1245	NULL
4	1245	INDEX_PAGE	0	1247	1243

View memory dump of non-clustered index leaf level page

Execute the following statement(s) to view memory dump of non-clustered index leaf level page (Replace 1243 in the below statement with **allocated_page_page_id** of a leaf level page [**Note**: Choose **allocated_page_page_id** with lowest value having **page_level '0'**])

```
-- Step 13: View non-clustered index leaf page
DBCC PAGE('SQLMaestros',1,1243,3);  -- Page ID will change in your case
GO
```



Editor | Results | Messages

	FileId	PageId	Row	Level	Column5 (key)	Column1 (key)	KeyHashValue	Row Size
1	1	1243	0	0	2014-03-04 13:47:12.200	1	(c369eed771c9)	16
2	1	1243	1	0	2014-03-04 13:47:12.217	2	(74af380810c8)	16
3	1	1243	2	0	2014-03-04 13:47:12.217	3	(8de3539ff5c4)	16
4	1	1243	3	0	2014-03-04 13:47:12.217	4	(b5c6641699b1)	16
5	1	1243	4	0	2014-03-04 13:47:12.217	5	(4c8a0f817cbd)	16
6	1	1243	5	0	2014-03-04 13:47:12.217	6	(acbe210eb801)	16
7	1	1243	6	0	2014-03-04 13:47:12.217	7	(55f24a995d0d)	16

Note: Since the clustered index does not contain any **UNIQUIFIER (key),** it's absent in non-clustered index as well and so will be the case with all non-clustered indexes on this table.

Cleanup	Execute the script provided in the cleanup section `------------------` `-- Begin: Cleanup` `------------------` `USE [master]` `GO` `ALTER DATABASE [SQLMaestros] SET  SINGLE_USER WITH ROLLBACK IMMEDIATE;` `GO` `DROP DATABASE [SQLMaestros];` `GO` `----------------` `-- End: Cleanup` `----------------`
Close all the query windows	Close all the query windows ( ) and if **SSMS** asks to save changes, click **NO**

Summary

In this exercise, you have learnt:

- Significance of Clustered Index Key
- Concept of UNIQUIFIER (key) in case of non-unique clustered index