Heap Internals

vLabs
by SQLMaestros

Table of Contents

Before You Begin

Estimated time to complete this lab

50 minutes

Objectives:

After completing this lab, we will be able to:

- Understand different types of allocation units in SQL Server
- Understand modification internals in a heap
- Understand how heap modifications can lead to excessive I/O

Prerequisites

Before working on this lab, you must have:

- Basic administration experience with SQL Server

Lab scenario

A heap is a table without a clustered index. One or more non-clustered indexes can be created on a heap. This lab is divided into three exercises. In the first exercise we will look at different types of allocation units available in SQL Server. In the second exercise we will observe how page modification is done in a heap and in the third exercise we will troubleshoot I/O issues related to page modification in heap.

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\Heap_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 Allocation Units in a Heap

Scenario

In this exercise, we will look at different types of allocation units in SQL Server.

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_AllocationUnit.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. Navigate to C:\vLabs\ 3. Select **1_AllocationUnit.sql** and click **Open**

Execute the statement(s) in the 'Setup' section	The setup section performs the following:
	• **SQLMaestros** database is created
	• **SQLMaestros** schema is created
	
	In **1_AllocationUnit.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 * FROM sys.databases WHERE name='SQLMaestros')
ALTER DATABASE [SQLMaestros] SET  SINGLE_USER WITH ROLLBACK IMMEDIATE;
DROP DATABASE [SQLMaestros];
GO
CREATE DATABASE SQLMaestros;
GO

USE SQLMaestros;
SET NOCOUNT ON;
GO

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

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

CREATE Table1 table

Execute the following statement(s) to **CREATE Table1** table with fixed length data types

Note: **INT**, **CHAR**, **DATETIME** etc. data types are considered as fixed length data types because irrespective of their actual usage they always take their maximum allocated space. For example if **CHAR** (1000) is defined during the table creation, irrespective of the size of the character string that is being inserted, 1000 bytes of space will be used.

```
-- Step 1: Create Table1 table in SQLMaestros database
CREATE TABLE [SQLMaestros].[Table1](
      Column1 INT,
      Column2 CHAR(9000));
GO
```

	Editor Messages Msg 131, Level 15, State 2, Line 4 The size (9000) given to the column 'Column2' exceeds the maximum allowed for any data type (8000). **Observation:** The above statement produces an error since maximum allowable size for any data type is 8000 bytes.

CREATE Table1 table	Execute the following statement(s) to **CREATE Table1** table with fixed length data type
	
	`-- Step 2: Create Table1 table in SQLMaestros database`
	`CREATE TABLE [SQLMaestros].[Table1](`
	`  Column1 INT,`
	`  Column2 CHAR(3000),`
	`  Column3 CHAR(3000),`
	`  Column4 CHAR(3000));`
	`GO`
	
	Editor Messages
	Msg 1701, Level 16, State 1, Line 2
	Creating or altering table 'Table1' failed because the minimum row size
	would be 9011, including 7 bytes of internal overhead. This exceeds the
	maximum allowable table row size of 8060 bytes.
	
	Observation: The above statement produces an error since maximum allowable row size for fixed length data types is 8060 bytes.

CREATE Table1 table	Execute the following statement(s) to **CREATE Table1** table with fixed length, variable length and LOB data types **Note**: **VARCHAR**, **NVARCHAR** etc. data types are considered as variable length data types because irrespective of their actual definition during the creation of the table, they always use space according to the usage. For example during the table definition if we have mentioned **VARCHAR**(1000) and during the insertion only five character is being inserted then actual space usage will be five bytes. Also note that there is some space overhead associated with variable data type's explanation for which is beyond the scope of this lab. **NTEXT**, **IMAGE** etc. data types are considered as LOB data type. `-- Step 3: Create Table1 table in SQLMaestros database`

```
CREATE TABLE [SQLMaestros].[Table1](
   Column1 INT,
   Column2 VARCHAR(3000),
   Column3 VARCHAR(3000),
   Column4 VARCHAR(3000),
   Column5 NTEXT);
GO
```

Note: For a variable length data type maximum allowable row size per row can exceed the 8060 bytes limit (Row Overflow).

View allocation details

Execute the following statement(s) to view allocation details for **Table1** table

```
-- Step 4: View allocation details for Table1 table
SELECT object_name(object_id) AS NAME
       ,partition_id
       ,partition_number AS pnum
       ,rows
       ,allocation_unit_id AS au_id
       ,type_desc AS page_type_desc
       ,total_pages AS pages
FROM sys.partitions p
INNER JOIN sys.allocation_units au ON p.partition_id = au.container_id
WHERE object_id = object_id('SQLMaestros.Table1');
GO
```

Editor | Results | Messages

	NAME	partition_id	pnum	rows	au_id	page_type_desc	pages
1	Table1	72057594039042048	1	0	72057594043564032	IN_ROW_DATA	0
2	Table1	72057594039042048	1	0	72057594043695104	LOB_DATA	0
3	Table1	72057594039042048	1	0	72057594043629568	ROW_OVERFLOW_DATA	0

Explanation: We have created **Table1** table with three different data types; fixed length (**INT**), variable length (**VARCHAR**) and LOB (**NTEXT**).

- **IN_ROW_DATA**: All data types except LOB data, max allowable size per row cannot exceed 8060 bytes

- **LOB_DATA**: Large object (LOB) data (**TEXT**, **NTEXT**, **IAMGE**, **XML**, large value types, and CLR user-defined types)
- **ROW_OVERFLOW_DATA**: (**VARCHAR**, **NVARCHAR**, **VARBINARY, sql_variant**, or CLR user-defined type columns). The length of each one of these columns must still remain within the limit of 8000 bytes; however, their combined widths can exceed the 8060-byte limit

INSERT a single record

Execute the following statement(s) to **INSERT** a single record in **Table1** table

Note: Value is supplied for **Column1** column only, rest are NULL.

```
-- Step 5: Insert a single fixed length record in Table1 table
INSERT INTO [SQLMaestros].[Table1] VALUES(1,NULL,NULL,NULL,NULL);
GO
```

View allocation details

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

```
-- Step 6: View data page and IAM page allocation due to the above insert
SELECT extent_file_id AS file_id
       ,allocated_page_page_id AS page_id
       ,page_type_desc AS page_type
       ,allocation_unit_type_desc AS allocation_unit
       ,extent_page_id
       ,allocated_page_iam_page_id AS iam_page_id
FROM sys.dm_db_database_page_allocations(DB_ID(), OBJECT_ID('SQLMaestros.Table1'), NULL, NULL, 'DETAILED');
GO
```

Editor | Results | Messages

	file_id	page_id	page_type	allocation_unit	extent_page_id	iam_page_id
1	1	296	IAM_PAGE	IN_ROW_DATA	296	NULL
2	1	287	DATA_PAGE	IN_ROW_DATA	280	296

	Observation: Column1 column data type is **INT,** which is a fixed length data type of 4 bytes. We are not inserting any data for **Column5** (LOB data type) column, thus in this case allocation unit(s) will be **IN_ROW_DATA.**
	
	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 }`
	`)`

INSERT a single record	Execute the following statement(s) to **INSERT** a single record in **Table1** table
	
	Note: Values are supplied for all columns except **Column5** (Not inserting any LOB data)
	
	`-- Step 7: Insert a single record in Table1 table`
	`DECLARE @data1 VARCHAR(3000)`
	`SET @data1  = REPLICATE('A',3000)`
	`INSERT INTO [SQLMaestros].[Table1] VALUES (2,@data1,@data1,@data1,NULL);`
	`GO`
	
	Note: Total row size for this insert is (4+3000+3000+3000) bytes or 9004 bytes, which is greater than maximum row size of 8060 bytes in **IN_ROW_DATA** allocation and we are not supplying any value for LOB data type, thus in this case **ROW_OVERFLOW_DATA** allocation will be done.

View allocation details	Execute the following statement(s) to view allocation details of **Table1** table

```
-- Step 8: View data page and IAM page allocation due to the above insert
SELECT extent_file_id AS file_id
       ,allocated_page_page_id AS page_id
       ,page_type_desc AS page_type
       ,allocation_unit_type_desc AS allocation_unit
       ,extent_page_id
       ,allocated_page_iam_page_id AS iam_page_id
FROM sys.dm_db_database_page_allocations(DB_ID(), OBJECT_ID('SQLMaestros.Table1'), NULL, NULL, 'DETAILED');
GO
```

Editor | Results | Messages

	file_id	page_id	page_type	allocation_unit	extent_page_id	iam_page_id
1	1	296	IAM_PAGE	IN_ROW_DATA	296	NULL
2	1	287	DATA_PAGE	IN_ROW_DATA	280	296
3	1	299	DATA_PAGE	IN_ROW_DATA	296	296
4	1	298	IAM_PAGE	ROW_OVERFLOW_DATA	296	NULL
5	1	297	TEXT_MIX_PAGE	ROW_OVERFLOW_DATA	296	298

Observation: Due to the above insert three new pages are allocated (299**,** 298, and 297), data page 299 with **IN_ROW_DATA** allocation and text mix page 297 with **ROW_OVERFLOW_DATA** allocation.

INSERT a single record	Execute the following statement(s) to **INSERT** a single record in **Table1** table

Note: Values are supplied for **Column1** and **Column5** columns (Inserting LOB data)

```
-- Step 9: Insert a single record of LOB data type in Table1 table
INSERT INTO [SQLMaestros].[Table1] VALUES (3,NULL,NULL,NULL,'TEXT');
GO
```

Note: Column5 column data type is **NTEXT** which is a LOB data type, thus in this case allocation will be **LOB_DATA** type.

View allocation details	Execute the following statement(s) to view allocation details for **Table1** table

```
-- Step 10: View data page and IAM page allocation due to the above insert
SELECT extent_file_id AS file_id
       ,allocated_page_page_id AS page_id
       ,page_type_desc AS page_type
       ,allocation_unit_type_desc AS allocation_unit
       ,extent_page_id
       ,allocated_page_iam_page_id AS iam_page_id
FROM sys.dm_db_database_page_allocations(DB_ID(), OBJECT_ID('SQLMaestros.Table1'), NULL, NULL, 'DETAILED');
GO
```

Editor | Results | Messages

	file_id	page_id	page_type	allocation_unit	extent_page_id	iam_page_id
1	1	296	IAM_PAGE	IN_ROW_DATA	296	NULL
2	1	287	DATA_PAGE	IN_ROW_DATA	280	296
3	1	299	DATA_PAGE	IN_ROW_DATA	296	296
4	1	301	IAM_PAGE	LOB_DATA	296	NULL
5	1	300	TEXT_MIX_PAGE	LOB_DATA	296	301
6	1	298	IAM_PAGE	ROW_OVERFLOW_DATA	296	NULL
7	1	297	TEXT_MIX_PAGE	ROW_OVERFLOW_DATA	296	298

Observation: New page (300) of **TEXT_MIX_PAGE** type is allocated with **LOB_DATA** allocation and an IAM page (301) to track this allocation.

Close all the query windows	Close all the query windows ( ) and if **SSMS** asks to save changes, click **NO**

Summary

In this exercise, we have learnt:

- Restrictions on column widths and row size
- How to view page allocations with sys.dm_db_database_page_allocations DMF
- About IN_ROW_DATA allocation
- About ROW_OVERFLOW_DATA allocation
- About LOB_DATA allocation

Exercise 2: Understanding Heap Page Modification Internals

Scenario

In this exercise, we will look at heap page modification due to update and we will also look at forwarding and forwarded record that occurs only in heap.

Tasks	Detailed Steps
Open **2_HeapModificationInternals.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. Navigate to C:\vLabs\ 3. Select **2_HeapModificationInternals.sql** and click **Open**

Execute the statement(s) in the 'Setup' section	The setup section performs the following:
	• **Table2** table is created with 3 records
	
	In **2_HeapModificationInternals.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(4000),
   Column3 VARCHAR(4000),
   Column4 CHAR(10));
GO

-- Insert 3 records in Table2 table
INSERT INTO [SQLMaestros].[Table2] VALUES(1,'smalldata','smalldata','smalldata');
INSERT INTO [SQLMaestros].[Table2] VALUES(2,'smalldata','smalldata','smalldata');
INSERT INTO [SQLMaestros].[Table2] VALUES(3,'smalldata','smalldata','smalldata');
GO

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

View data page allocation

Execute the following statement(s) to view data page allocated to **Table2** table due to above **INSERT**

```
-- Step 1: Data pages allocated to table Table2 due to above insert
SELECT allocated_page_page_id
       ,page_type_desc
       ,*
FROM sys.dm_db_database_page_allocations(DB_ID(), OBJECT_ID('SQLMaestros.Table2'), NULL, NULL,
'DETAILED')
WHERE page_type = 1;
GO
```

Editor | Results | Messages

	allocated_page_page_id	page_type_desc	database_id	object_id	index_id	pa
1	302	DATA_PAGE	17	949578421	0	1

View data page memory dump	Execute the following statement(s) to view memory dump of data page of **Table2** table (Replace 302 in the below statement with the **allocated_page_page_id** we got from step 1)

```
-- Step 2: View data page memory dump
DBCC TRACEON(3604);
DBCC PAGE('SQLMaestros',1,302,3); --Page ID will change in your case
GO
```

Observation: In the **PAGE HEADER** section of the above DBCC output **m_freeCnt** is showing 7958, which means the data page has 7958 bytes of free space. Currently all the three records are there in this single page.

```
PAGE HEADER:

m_pageId = (1:302)                  m_headerVersion = 1                 m_type = 1
m_typeFlagBits = 0x0                m_level = 0                         m_flagBits = 0x8000
m_objId (AllocUnitId.idObj) = 89    m_indexId (AllocUnitId.idInd) = 256
Metadata: AllocUnitId = 72057594043760640
Metadata: PartitionId = 72057594039107584                                Metadata: IndexId = 0
Metadata: ObjectId = 949578421      m_prevPage = (0:0)                  m_nextPage = (0:0)
pminlen = 17                        m_slotCnt = 6                       m_freeCnt = 7820
m_freeData = 360                    m_reservedCnt = 0                   m_lsn = (37:164:2)
m_xactReserved = 0                  m_xdesId = (0:0)                    m_ghostRecCnt = 0
m_tornBits = 0                      DB Frag ID = 1
```

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}
       )
```

UPDATE a single record

Execute the following statement(s) to **UPDATE** a single record(WHERE COL1 = 2) in **Table2** table

```
-- Step 3: Update a single record in Table2 table
DECLARE @DATA VARCHAR(4000)
SET @DATA = REPLICATE('bigdata',570);
UPDATE [SQLMaestros].[Table2] SET Column2 = @DATA, Column3 = @DATA WHERE Column1 = 2;
GO
```

Explanation: A page can contain a maximum of 8060 bytes of data. In the previous task we have observe that **TABLE2** data page has 7958 bytes of free space. We are updating one record with an additional 8000 bytes of data. In order to perform this **UPDATE** SQL Server will create a new page that will hold this updated data (**FORWARDED RECORD**). And the existing page where the original record was will now hold a pointer to this new page (**FORWARDING_STUB**).

View data page allocation

Execute the following statement(s) to view data page allocation for **Table2** table due to the above **UPDATE**

```
-- Step 4: View sata pages allocated after above update
SELECT allocated_page_page_id
       ,page_type_desc
       ,*
FROM sys.dm_db_database_page_allocations(DB_ID(), OBJECT_ID('SQLMaestros.Table2'), NULL, NULL,
'DETAILED')
WHERE page_type = 1;
GO
```

Editor | Results | Messages

	allocated_page_page_id	page_type_desc	database_id	object_id	inde
1	302	DATA_PAGE	17	981578535	0
2	304	DATA_PAGE	17	981578535	0

Observation: Before **UPDATE** operation we have a single data page (302), and after update a new data page (304) is allocated.

View memory dump of the old page	Execute the following statement(s) to view memory dump of the old page (302) in **Table2** table (Replace 302 in the below command with **allocated_page_page_id** we got in step 1)
	
	`-- Step 5: View old page memory dump`
	`DBCC PAGE('SQLMaestros',1,302,3); --Page ID will change in your case`
	`GO`
	
	`Record Type = FORWARDING_STUB  Record Attributes =  Record Size = 9`
	
	`Memory Dump @0x000000006D07A08D`
	
	`0000000000000000:  04300100 00010000 00  .0.....`
	`Forwarding to  =  file 1 page 304 slot 0`

View memory dump of the new page	Execute the following statement(s) to view memory dump of the new page (304) in **Table2** table (Replace 304 in the below statement with **page_id** we get in the output of step 5 'Forwarding to' segment)
	
	`-- Step 6: View new page memory dump`
	`DBCC PAGE('SQLMaestros',1,304,3); --Page ID will change in your case`
	`GO`
	
	`Record Type = FORWARDED_RECORD  Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS`
	`Record Size = 8019`
	`Memory Dump @0x000000006E3AA060`
	
	`0000000000000000:  32001200 02000000 736d616c 6c646174 61200400  2.....smalldata ..`
	`0000000000000014:  000300b3 0f491f53 9f626967 64617461 62696764  ...³.I.Sbigdatabigd`
	`0000000000000028:  61746162 69676461 74616269 67646174 61626967  atabigdatabigdatabig`
	`.....`
	`.....`
	
	`Forwarded from  =  file 1 page 302 slot 1`

Close all the query windows	Close all the query windows ( ) and if **SSMS** asks to save changes, click **NO**

Summary

In this exercise, we have learnt:

- Concept of heap page modification.
- Forwarding and forwarded record in heap.
- How to view a page contents using DBCC PAGE command
- How to use sys.dm_db_database_page_allocations DMF to view page information

Exercise 3: Troubleshooting Heap Page Modifications

Scenario

In this exercise, we will look at performance issues caused by heap page modification and how to troubleshoot issues caused by heap page modification.

Tasks	Detailed Steps
Open **3_TroubleshootingHeap.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. Navigate to C:\vLabs\ 3. Select **3_TroubleshootingHeap.sql** and click **Open**

Execute the statement(s) in the 'Setup' section	The setup section performs the following:
	• **Table2** table is created with 1000 records
	
	In **3_TroubleshootingHeap.sql**, Review and execute the statement(s) 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 VARCHAR(4000),`
	`  Column3 VARCHAR(4000),`
	`  Column4 CHAR(9))`

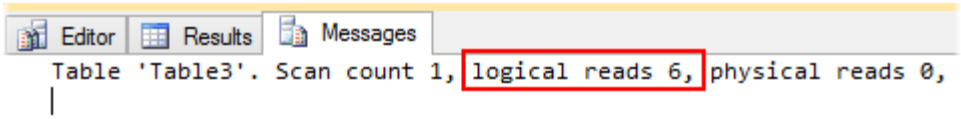
```
GO

-- Insert 1000 records in Table3 table
DECLARE @COUNT INT;
SET @COUNT = 1;
WHILE @COUNT < 1001
BEGIN
	INSERT INTO [SQLMaestros].[Table3]
	VALUES (
		@COUNT
		,'smalldata'
		,'smalldata'
		,'smalldata'
		);
	SET @COUNT = @COUNT + 1;
END

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

Execute a **SELECT** statement

Execute the following statement(s) to **SELECT** data from **Table3** table

Note: We have enabled statistics i/o by **SET STATISTICS IO ON** which will display I/O statistics for the workload.

```
-- Step 1: Execute a select statement on Table3 table
SET STATISTICS IO ON;
SELECT * FROM [SQLMaestros].[Table3];
GO
```



Observation: Currently the query is requiring six logical reads. In the next section we will explain this.

View table data size	Execute the following statement(s) to view data size of **Table3** table
	
	`-- Step 2: Execute the following system stored procedure to find size of data in Table3 table`
	`EXEC sp_spaceused 'SQLMaestros.Table3';`
	`GO`
	
	Editor
		Results	Messages
			
			| | name | rows | reserved | data | index_size | unused |
			| 1 | Table3 | 1000 | 56 KB | 48 KB | 8 KB | 0 KB |
			
			Observation: A data page size is 8 KB and the table size is 48 KB. Thus there must be 48/8 = 6 data pages in **Table3** table to accommodate the data. That's the reason why we get 6 logical reads in the above **SELECT** statement.

Execute an **UPDATE** statement	Execute the following statement(s) to **UPDATE Column2** and **Column3** columns of **Table3** table
	
	`-- Step 3: Update records in Column2 and Column3 of Table3`
	`DECLARE @DATA VARCHAR(4000)`
	`SET @DATA = REPLICATE('bigdata',570);`
	`UPDATE [SQLMaestros].[Table3] SET Column2 = @DATA, Column3 = @DATA;`
	`GO`
	
	Explanation: A page can contain a maximum of 8060 bytes of data. Here we are updating all the records with an additional 8000 bytes of data per row. In order to perform this **UPDATE**, SQL Server will allocate new pages that will hold this updated data (**FORWARDED RECORD**). And the existing pages that contained the original records will now hold a pointer to the new pages (**FORWARDING_STUB**).

Execute a **SELECT** statement	Execute the following statement(s) to **SELECT** data from **Table3** table
	
	`-- Step 4: Execute a select query on Table3 table`
	`SELECT * FROM [SQLMaestros].[Table3];`
	`GO`

	Editor | Results | Messages `Table 'Table3'. Scan count 1, logical reads 2006, physical reads 0,` **Observation:** The same **SELECT** query which earlier required 6 logical reads is now taking 2006 logical reads after the **UPDATE** operation. However, the data size is not that much which we will explain in the next section.

View table data size	Execute the following statement(s) to view data size of **Table3** table
	
	`-- Step 5: Execute the following system stored procedure to find size of data in table Table3`
	`EXEC sp_spaceused 'SQLMaestros.Table3';`
	`GO`
	
	Editor | Results | Messages
	| | name | rows | reserved | data | index_size | unused |
	| 1 | Table3 | 1000 | 8072 KB | 8048 KB | 8 KB | 16 KB |
	
	Observation: A data page is of 8 KB size and the table size is 8048 KB. Thus there must be 8048/8 = 1006 data pages in **Table3** table to accommodate this data. But the **SELECT** statement requires 2006 logical reads instead of 1006 logical reads because we have additional 1000 forwarded records due to heap modification. To confirm this, we will look at the no. of forwarded records, in the next step

View number of forwarded records	Execute the following statement(s) to view number of forwarded records for **Table3** table
	
	`-- Step 6: Get forwarded record count`
	`SELECT index_type_desc`
	`  ,forwarded_record_count`
	`FROM sys.dm_db_index_physical_stats(DB_ID(N'SQLMaestros'), OBJECT_ID(N'SQLMaestros.Table3'), NULL, NULL, 'DETAILED');`
	`GO`

Editor | Results | Messages

	index_type_desc	forwarded_record_count
1	HEAP	1000

Note: We can use **sys.dm_db_index_physical_stats()** DMF to get index details. 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 }
       )
```

CREATE a clustered index

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

```
-- Step 7: Create a clustered index on Column1 of Table3 and then drop it
CREATE CLUSTERED INDEX CL_Table3_Column1 ON [SQLMaestros].[Table3](Column1);
GO
DROP INDEX CL_Table3_Column1 ON [SQLMaestros].[Table3];
GO
```

Explanation: Creating and dropping the clustered index re-arranges the data pages in the order of clustering key thereby removing the **FORWARDED_RECORD** and **FORWRDING_STUB.**

Execute a **SELECT** statement	Execute the following statement(s) to **SELECT** data from **Table3** table
	
	`-- Step 8: Execute a select query on Table3`
	`SELECT * FROM [SQLMaestros].[Table3];`
	`GO`
	
	Editor
		Results	Messages
			`Table 'Table3'. Scan count 1, logical reads 1000, physical reads 0,`
			
			Observation: As we have removed all the forwarded records from **Table3** table, the same **SELECT** statement is now taking 1000 logical reads.

View table data size	Execute the following statement(s) to view data size of **Table3** table `-- Step 9: Execute the following system stored procedure to find size of data in table Table3` `EXEC sp_spaceused 'SQLMaestros.Table3';` `GO` Editor	Results	Messages |	name	rows	reserved	data	index_size	unused | | 1	Table3	1000	8008 KB	8000 KB	8 KB	0 KB | **Observation:** A data page is of 8 KB size and the table size is 8000 KB. Thus there must be 8000/8 = 1000 data pages in **Table3** table to accommodate this data; hence we are getting 1000 logical reads for the above **SELECT** statement.

Cleanup	Execute the following script in 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, we have learnt:

- How heap modifications can lead to extra data pages (forwarded records).
- Using sp_spaceused system stored procedure to find the data size of a table.
- Troubleshooting heap modification by creating a clustered index and then dropping it.
- Concept of logical reads and how it's directly related to the table data size and heap modification.