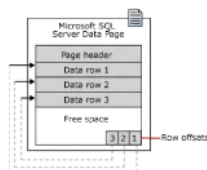
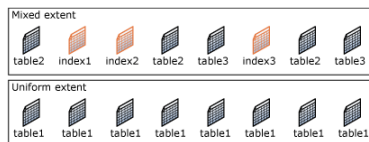


In SQL Server, the **page size is 8-KB**. This means SQL Server databases have **128 pages per megabyte**. Each page begins with a **96-byte header** that is used to store system information about the page. This information includes the **page number, page type, the amount of free space on the page**, and the allocation unit ID of the object that owns the page.

From <https://docs.microsoft.com/en-us/sql/relational-databases/pages-and-extends-architecture-guide?view=sql-server-2017>



- Uniform extents are owned by a single object; all eight pages in the extent can only be used by the owning object.
- Mixed extents are shared by up to eight objects. Each of the eight pages in the extent can be owned by a different object.



```

1 Use Northwind;
2 Go
3
4 -- Examine data page allocations in the following tables using the
5 -- new DMF sys.dm_db_database_page_allocations
6 SELECT *
7 FROM sys.dm_db_database_page_allocations(db_id('Northwind'),
8 object_id('Employees'), 1, null, 'DETAILED')
9 WHERE page_type_desc = 'DATA_PAGE';
10
11 SELECT *
12 FROM sys.dm_db_database_page_allocations(db_id('Northwind'),
13 object_id('Customers'), 1, null, 'DETAILED')
14 WHERE page_type_desc = 'DATA_PAGE'
15 Order By extent_page_id;
16
17 SELECT *
18 FROM sys.dm_db_database_page_allocations(db_id('Northwind'),
19 object_id('Order_Details'), 1, null, 'DETAILED')
20 WHERE page_type_desc = 'DATA_PAGE'
21 Order By extent_page_id;
22

```

**Results Messages**

	allocation_unit_type	allocation_unit_type_desc	data_file_id	clone_state	clone_state_desc	extent_file_id	extent_page_id	allocated_page_page_id	allocated_page_page_id	is_allocated	n_nem_page	n_need_page_allocation	page_free_space_percent	page_type	page_type_desc	page_level	next_page_file_id	next_page_page_id	previous_page_file_id	previous_page_page_id	n_page_count
1	DWG	IN_ROW_DATA	NULL	NULL	SECONDARY	1	448	1	448	1	0	0	NULL	1	DATA_PAGE	0	NULL	NULL	NULL	NULL	0
2	1120	IN_ROW_DATA	NULL	NULL	SECONDARY	1	400	1	378	1	0	0	NULL	1	DATA_PAGE	0	1	402	NULL	NULL	0
3	1120	IN_ROW_DATA	NULL	NULL	SECONDARY	1	400	1	378	1	0	0	NULL	1	DATA_PAGE	0	NULL	NULL	1	400	0

Query executed successfully.

(local) [11.0.501] LABGURU-OFFICE\dele (55) : Northwind (00:00:00 208)

## Contents

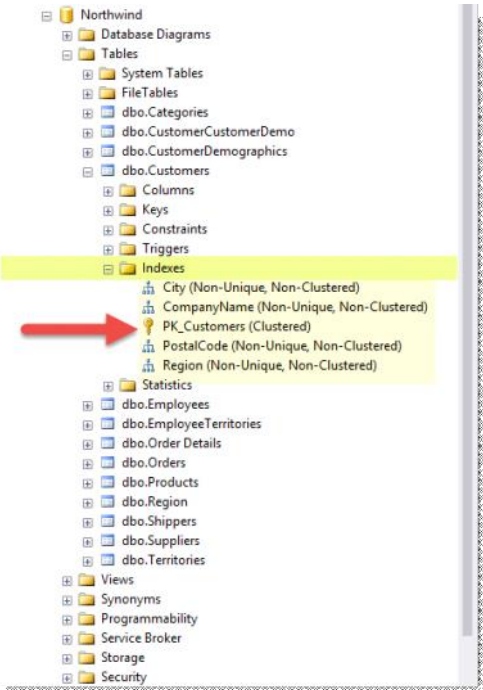
LAB 1 – Install Data for Entities	4
LAB 2 – Installing Entity Framework	8
LAB 3 – Querying Data	23
LAB 4 – Entity Framework Data Layer	28
LAB 5 – Change Database and Update Model	37
LAB 6 – Entity Framework Enums	51
LAB 8 – LINQ Data Techniques	57
LAB 9 – LINQ Manipulation	72
LAB 10 – Entity Framework Model First	85
LAB 11 – Entity Framework Code First	109
LAB 12 – Code	124
LAB 13 – Data Annotations / Database	137
LAB 14 – Data Annotation Validation	147
LAB 15 – Data Annotation Formatting with MVC	164
LAB 16 – Customizing the DbContext & DbSet	178
LAB 17 – Lazy / Eager / Explicit Loading	191
Appendix A – Download Files from Labs.Guru	201

## Contents

LAB 1 – jQuery Ready Event	5
LAB 2 – Text vs Value	11
LAB 3 – Counting Elements	15
LAB 4 – Selectors	18
LAB 5 – Using CSS Classes	24
LAB 6 – Using Attributes	28
LAB 7 – Checkbox and Select Elements	32
LAB 8 – Collection Search with (Contains)	37
LAB 9 – Using the Select Control	42
LAB 10 – Using Before & After	47
LAB 11 – Appending Content	51
LAB 12 – Rewrite Document Dynamically	54
LAB 13 – Wrapping Content	57
LAB 14 – Event Binding with jQuery	61
LAB 15 – Event Arguments with jQuery	66
LAB 16 – Mouse Events and Position	72
LAB 17 – Hover / Rollovers	77
LAB 18 – Keystroke Capturing	81
LAB 19 – Controlling the Tab Keystroke on a Document	86
LAB 20 – Slide Toggle	90
LAB 21 – Custom Toggle for Effects	93
LAB 22 – Advanced Effects	97

Data Index #2

Tuesday, May 23, 2017 11:10 AM



Impacts on Searches

- **Selectivity**
  - A measure of how many rows are returned compared to the total number of rows
  - **High selectivity** means a **small number of rows** when related to the total number of rows
- **Density**
  - A measure of the lack of uniqueness of data in the table
  - **High density** indicates a **large number of duplicates**
- **Index Depth**
  - **Number of levels within the index**
  - Common misconception that indexes are deep

Figure 1.1 Index Leaf Nodes and Corresponding Table Data

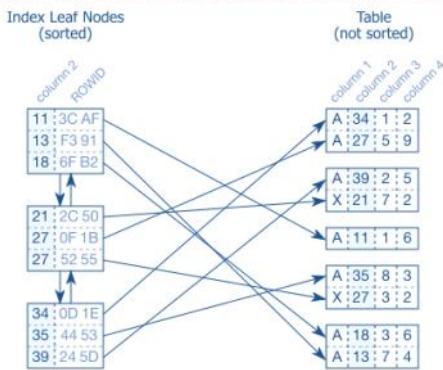
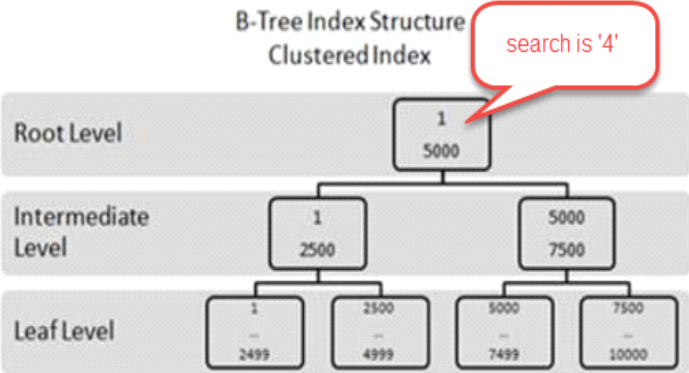


Figure 1.1 illustrates the index leaf nodes and their connection to the table data. Each index entry consists of the indexed columns (the key, column 2) and refers to the corresponding table row (via ROWID or RID).

B-Tree Index Structure  
Clustered Index



Customer Number	First Name	Last Name
4	Anna	Victoria

# HEAP Versus Clustered

Tuesday, May 23, 2017 6:09 PM

An **Heap** is good for small Data or Data that is NOT Queried Often

## HEAP

- Data is not stored in any particular order
- Specific data can not be retrieved quickly, unless there are also non-clustered indexes
- Data pages are not linked, so sequential access needs to refer back to the index allocation map (IAM) pages
- Since there is no clustered index, additional time is not needed to maintain the index
- Since there is no clustered index, there is not the need for additional space to store the clustered index tree
- These tables have a index\_id value of 0 in the sys.indexes catalog view

## Clustered Table

- Data is stored in order based on the clustered index key
- Data can be retrieved quickly based on the clustered index key, if the query uses the indexed columns
- Data pages are linked for faster sequential access
- Additional time is needed to maintain clustered index based on INSERTS, UPDATES and DELETES
- Additional space is needed to store clustered index tree
- These tables have a index\_id value of 1 in the sys.indexes catalog view

Send **HeapVsClustered.sql**

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the Object Explorer displays the 'Northwind' database structure, including tables like 'dbo.Categories', 'dbo.CustomerCustomerDemo', 'dbo.CustomerDemographics', 'dbo.Customers', 'dbo.Employees', 'dbo.EmployeeTerritories', 'dbo.Order Details', 'dbo.Orders', 'dbo.Products', 'dbo.Region', 'dbo.Shippers', 'dbo.Suppliers', and 'dbo.Territories'. A red arrow points from the 'dbo.Territories' table in the Object Explorer to the 'Results' pane on the right. The 'Results' pane displays the output of the 'HeapVsClustered.sql' query, which lists the table names and their storage type (Clustered or Heap).

(No column name)	(No column name)
1	dbo.sysdiagrams Clustered
2	dbo.CustomerDemographics Heap
3	dbo.Region Heap
4	dbo.Employees Clustered
5	dbo.Categories Clustered
6	dbo.Customers Clustered
7	dbo.Shippers Clustered
8	dbo.Suppliers Clustered
9	dbo.Orders Clustered
10	dbo.Products Clustered
11	dbo.Order Details Clustered
12	dbo.CustomerCustomerDemo Heap
13	dbo.Territories Heap
14	dbo.EmployeeTerritories Heap

Files: **PersonHeapForwardPointers.sql**

Student File: **spForwardPointers.sql**

## Create Heap Table

**RUN EACH STEP SEPARATLY**

```

PersonHeapForwardP... (LABS\delip (55)) * X HeapVsClustered.sql (LABS\delip (53))
3
4 --RUN EACH STEP SEPARATELY
5
6 --Step One
7
8 CREATE TABLE PersonHeap
9 (
10     PersonID uniqueidentifier DEFAULT NEWID(),
11     FirstName VarChar(25) NOT NULL,
12     LastName VarChar(25) NOT NULL,
13     Address VarChar(50) NULL,
14     City VarChar(25) NULL,
15     State VarChar(2) NOT NULL,
16     ZipCode VarChar(25) NULL,
17 )
18 GO
19 Exec spForwardPointers 'PersonHeap';
20
21 --Step Two
22
23 INSERT INTO PersonHeap VALUES
24 (NewID(), 'Mickey', 'Mouse', '123 Main Street', 'Sacramento', 'CA', '95830');
25 GO 1000
26 Select * From PersonHeap
27
28 --Step Three
29
30 UPDATE PersonHeap
31 SET Address = '123456789012345678901234567890', City= '1234567890123456789012345
32 WHERE LastName = 'Mouse';
33 GO
34 Select * From PersonHeap
35 Exec spForwardPointers 'PersonHeap';
36
37
38 --Step Four / Fix Forward Pointers
39
40 Alter table PersonHeap rebuild
41 Exec spForwardPointers 'PersonHeap';
42
43 Drop Table PersonHeap

```

Database_id	Index_id	TableName	IndexName	Index_type_desc	Avg_fragmentation_in_percent	forwarded_record_count
1	10	PersonHeap	HEAP	HEAP	0	0

Database_id	Index_id	TableName	IndexName	Index_type_desc	Avg_fragmentation_in_percent	forwarded_record_count
1	10	PersonHeap	HEAP	HEAP	0	265

Query executed successfully. (local)\sqlservr\13.0... LABS\delip (55) Schools 00:00:06 2001 rows

Database_id	Index_id	TableName	IndexName	Index_type_desc	Avg_fragmentation_in_percent	forwarded_record_count
1	10	PersonHeap	HEAP	HEAP	0	0

```

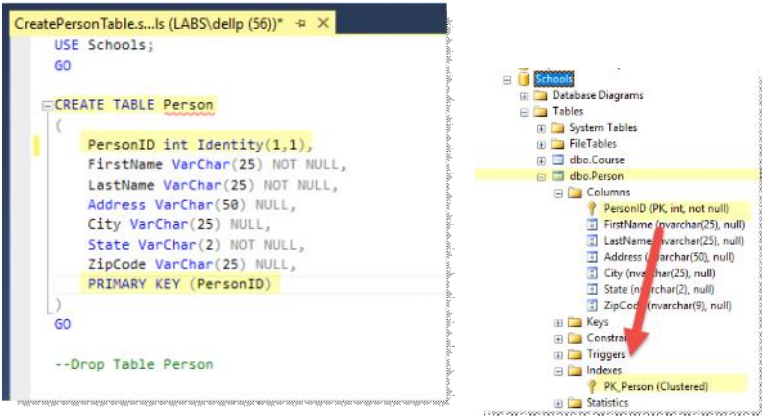
spForwardPointers.s... (LABS\delip (51)) * X PersonHeapForwardP... (LABS\delip (55)) *
1 Use Schools;
2 Go
3
4 Create Procedure spForwardPointers
5     @Table VarChar(15)
6 AS
7
8 --Check Forward Pointer
9
10 Select Database_id,
11     Index_id,
12     Object_name([object_id]) as TableName,
13     Case when SI.name is null then 'HEAP' else SI.name End as IndexName,
14     Index_type_desc,
15     Avg_fragmentation_in_percent,
16     forwarded_record_count
17 From sys.dm_db_index_physical_stats(db_id(),object_id(@Table),null,null,'det
18 Inner join sys.sysindexes AS SI on SDDIPS.[object_id] = SI.id
19 AND SDDIPS.index_id = SI.index_id
20 Where index_level = 0
21 Go
22

```

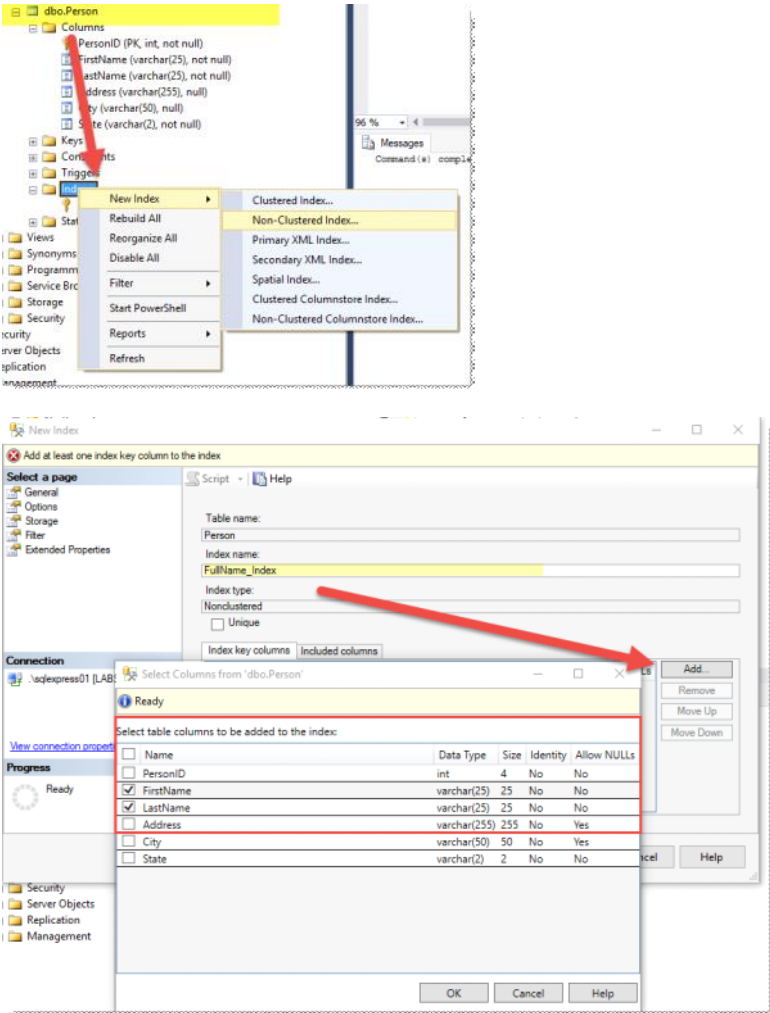


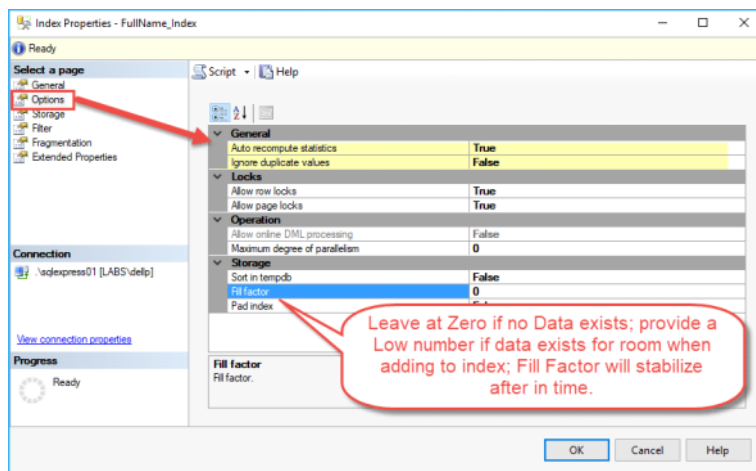
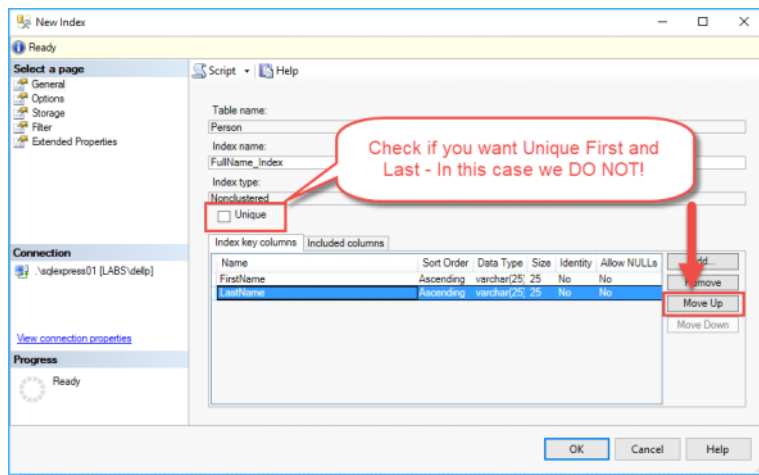
CreatePerson.Sql

- ☐ Create Table Person With Primary Key (PersonID Identity)
- ☐ Check Index on PersonID



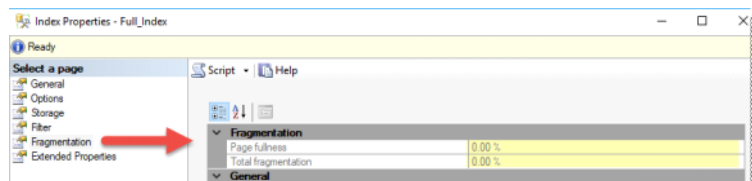
- ☐ Create Name Index
- ☐ Check Fragmentation (Properties)





Index key columns		Included columns				
Name	Sort Order	Data Type	Size	Identity	Allow NULLs	
LastName	Ascending	varchar(25)	25	No	No	
FirstName	Ascending	varchar(25)	25	No	No	

Create Index, then come back and check current Fragmentation



# Disabling an Index

Tuesday, May 23, 2017 7:45 PM

## ☐ Disabling an Index

```
ALTER INDEX FullName_Index ON Person DISABLE;  
GO
```

## ☐ Re-Enable DISABLED Index

```
ALTER INDEX FullName_Index ON Person REBUILD;  
GO
```

## ☐ Dropping an Index

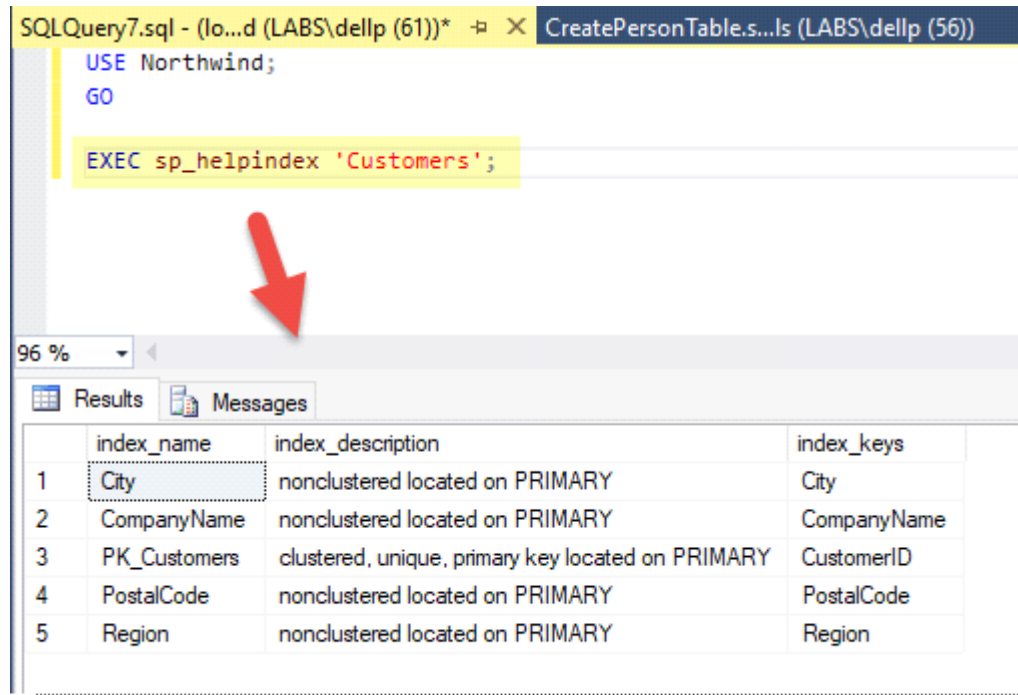
```
DROP INDEX FullName_Index ON Person;  
GO
```



# Find Indexes (Table)

Tuesday, May 23, 2017 8:04 PM

## FindIndexes.sql



The screenshot shows a SQL query window with the following text:

```
USE Northwind;  
GO  
EXEC sp_helpindex 'Customers';
```

A red arrow points from the query text to the Results tab. The Results tab displays the following table:

	index_name	index_description	index_keys
1	City	nonclustered located on PRIMARY	City
2	CompanyName	nonclustered located on PRIMARY	CompanyName
3	PK_Customers	clustered, unique, primary key located on PRIMARY	CustomerID
4	PostalCode	nonclustered located on PRIMARY	PostalCode
5	Region	nonclustered located on PRIMARY	Region

# Script Clustered Index

Tuesday, May 23, 2017 8:11 PM

## One Clustered Index / Table

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the Object Explorer shows the database structure for 'Schools'. The 'Tables' folder is expanded, and 'EmployeeID\_Index (Clustered)' is highlighted. A red arrow points from the 'Indexes' folder to this specific index. On the right, the SQL Query window shows the following script:

```
USE Schools;
GO

-- Create a new table with three columns.
CREATE TABLE dbo.EmployeeClustered
(
    EmployeeID UniqueIdentifier Default NewID(),
    FirstName nvarchar(25) NULL,
    LastName nvarchar(25) NULL,
    HireDate datetime);
GO

-- Create a clustered index
CREATE CLUSTERED INDEX EmployeeID_Index
ON dbo.EmployeeClustered (EmployeeID);
GO
```

At the bottom, the Messages pane indicates that the command(s) completed successfully.

## Script Non-Clustered Index

Tuesday, May 23, 2017 8:14 PM

### ☐ Put If statements in after Table and Index are created...

The **FILLFACTOR** option for an index determines the percentage of free space that is reserved on each leaf level page of an index when it is created or rebuilt  
From <[https://dbamohsin.wordpress.com/tag/pad\\_index/](https://dbamohsin.wordpress.com/tag/pad_index/)>

**PAD\_INDEX=ON**; This option specifies index padding. When turned ON, it uses the percentage specified by FILLFACTOR is applied to the intermediate-level and root level pages of an index.

From <[https://dbamohsin.wordpress.com/tag/pad\\_index/](https://dbamohsin.wordpress.com/tag/pad_index/)>

The screenshot displays the SQL Server Enterprise Manager interface on the left and a script window on the right. The Enterprise Manager shows the database structure for 'LABS\dellp', with a red arrow pointing to the 'FullName\_Index (Non-Unique, Non-Clustered)' under the 'Indexes' folder. The script window, titled 'CreateNonClustered...s (LABS\dellp (63))', contains the following T-SQL code:

```
USE Schools;
GO

IF EXISTS (SELECT name FROM sys.indexes
           WHERE name = N'FullName_Index')
DROP INDEX FullName_Index ON EmployeeNonClustered;
GO

IF EXISTS (SELECT name FROM sys.tables
           WHERE name = N'EmployeeNonClustered')
DROP TABLE EmployeeNonClustered;
GO

-- Create a new table with three columns.
CREATE TABLE dbo.EmployeeNonClustered
(
    EmployeeID UniqueIdentifier Default NewID(),
    FirstName nvarchar(25) NULL,
    LastName nvarchar(25) NULL,
    HireDate datetime);
GO

-- Create a non-clustered index
CREATE NONCLUSTERED INDEX FullName_Index
ON dbo.EmployeeNonClustered
(
    LastName ASC,
    FirstName ASC
) With (Pad_Index=OFF, FillFactor=70); --Page 4-18
GO
```

Annotations in the script window include:

- Red circle 1: Points to the `CREATE TABLE` statement.
- Red circle 2: Points to the `CREATE NONCLUSTERED INDEX` statement.
- Red circle 3: Points to the `IF EXISTS` block that checks for the existence of the index and table before dropping them.

## Create Person - *CreatePersonTable.sql*

```

CreateIndexFullNam...s (LABS\delip (67))  CreatePersonTable.s...ls (LABS\delip (56))
USE Schools;
GO

CREATE TABLE Person
(
    PersonID uniqueidentifier DEFAULT NEWID(),
    FirstName VarChar(25) NOT NULL,
    LastName VarChar(25) NOT NULL,
    Address VarChar(50) NULL,
    City VarChar(25) NULL,
    State VarChar(2) NOT NULL,
    ZipCode VarChar(25) NULL,
    PRIMARY KEY (PersonID)
)
GO
    
```

## Create FullName\_Index

### CreateIndexFullName.sql

```

CreateIndexFullNam...s (LABS\delip (67))*  CreatePersonTable.s...ls (LABS\delip (56))
USE [Schools]
GO

CREATE NONCLUSTERED INDEX [FullName_Index] ON [dbo].[Person]
(
    [LastName] ASC,
    [FirstName] ASC
)
GO
    
```

## Add Rows

## Check Fragmentation

### File: Insert 1000Rows

```

CreatePersonTable.s...ls (LABS\delip (53))  Insert 1000 Rows.sql...ls (LABS\delip (61))*
USE Schools
Go

INSERT INTO Person VALUES
(NewID(), 'Mickey', 'Mouse', '123 Main Street', 'Sacramento', 'CA', '95838')
GO 1000

--Drop Index FullName_Index;
--Drop Table Person;
    
```

SQLQuery7.sql - (lo...ls (LABS\delip (54)) Insert 1000 Rows.sql...ls (LABS\delip (61))\*

```

/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [PersonID]
, [FirstName]
, [LastName]
, [Address]
, [City]
, [State]
FROM [Schools].[dbo].[Person]
    
```

96 %

PersonID	FirstName	LastName	Address	City	State
1	Mickey	Mouse	123 Main Street	Sacramento	CA
2	Mickey	Mouse	123 Main Street	Sacramento	CA

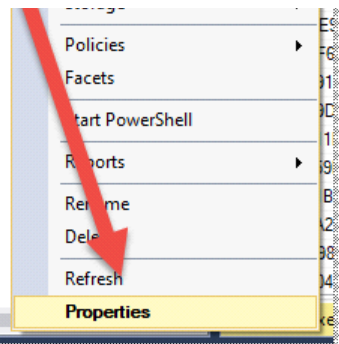
Indexes

- PK\_Person\_AA2FFB85D5E3E2D4 (PK)
- FullName\_Index (Non-Unique, Non-Clustered)

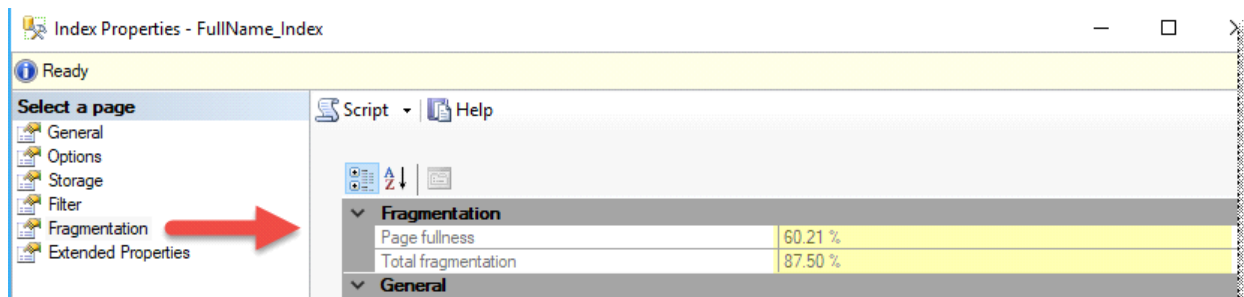
Right-click context menu for FullName\_Index:

- New Index
- Script Index as
- Rebuild
- Reorganize
- Disable
- Storage
- Policies
- Facets

	PersonID	FirstName	LastName	Address	City	State	Objects ition ement
1	1	Mickey	Mouse	123 Main Street	Sacramento	CA	
2	2	Mickey	Mouse	123 Main Street	Sacramento	CA	
3	3	Mickey	Mouse	123 Main Street	Sacramento	CA	
4	4	Mickey	Mouse	123 Main Street	Sacramento	CA	
5	5	Mickey	Mouse	123 Main Street	Sacramento	CA	
6	6	Mickey	Mouse	123 Main Street	Sacramento	CA	
7	7	Mickey	Mouse	123 Main Street	Sacramento	CA	
8	8	Mickey	Mouse	123 Main Street	Sacramento	CA	
9	9	Mickey	Mouse	123 Main Street	Sacramento	CA	
10	10	Mickey	Mouse	123 Main Street	Sacramento	CA	
11	11	Mickey	Mouse	123 Main Street	Sacramento	CA	
12	12	Mickey	Mouse	123 Main Street	Sacramento	CA	



## CHECK FRAGMENTATION



## CLEAN FRAGMENTATION - REBUILD

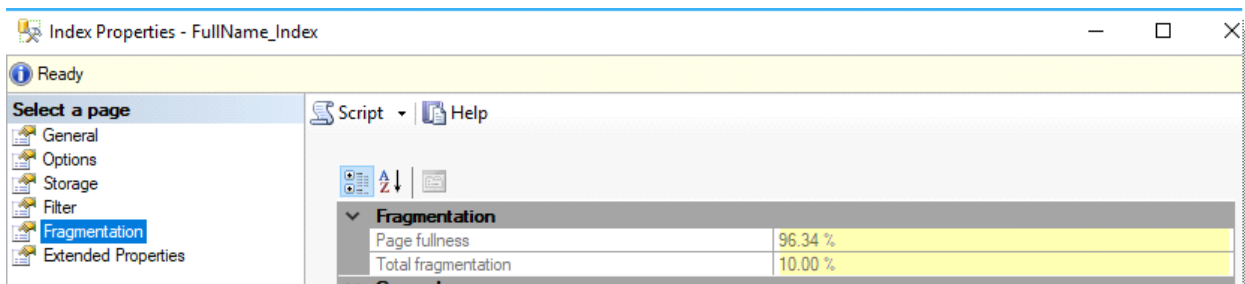
### CleanIndexFragmentationRebuild.sql

```

CleanIndexFragment...s (LABS\delip (58))  CreatePersonTable
USE Schools;
GO

--Fix Fragment Allocation Issue
ALTER INDEX FullName_Index ON Person REBUILD

```



## OR - TO CLEAN FRAGMENTATION - DROP/RE-CREATE

### ClearIndexFragmentationDropCreate.sql

```
CleanIndexFragment...s (LABS\delip (67))  X CleanIndexFragment...s (LABS\delip (58))
USE [Schools]
GO

DROP INDEX FullName_Index ON Person;
/***** Object: Index [FullName_Index]    Script Date: 5/23/2017 7:30:45 PM *****/
CREATE NONCLUSTERED INDEX [FullName_Index] ON [dbo].[Person]
(
    [LastName] ASC,
    [FirstName] ASC
)ON [PRIMARY]
GO
```

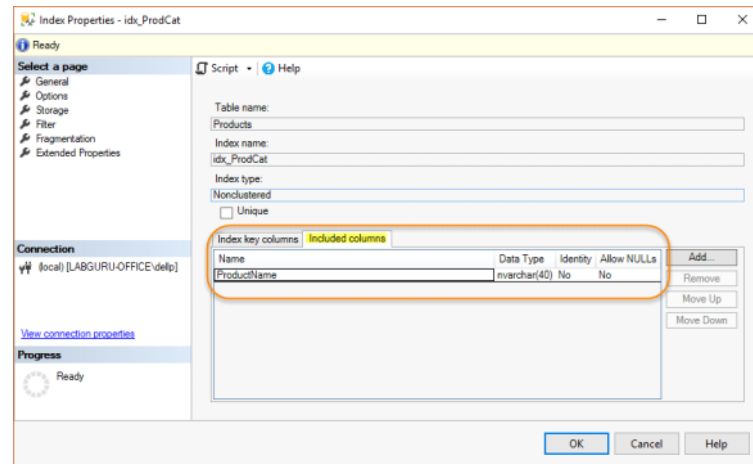


# Include And Filtered Index

Sunday, December 3, 2017 3:14 PM

## Include Index

```
16 CREATE NONCLUSTERED INDEX idx_ProdCat
17 ON Products(ProductID, CategoryID)
18 INCLUDE (ProductName); --Covered w/o sorting
19
20 -- index index Retrieve but not sort
21 SELECT ProductID, CategoryID, ProductName
22 FROM Products
23 WHERE CategoryID = 5;
```



## Filtered Index

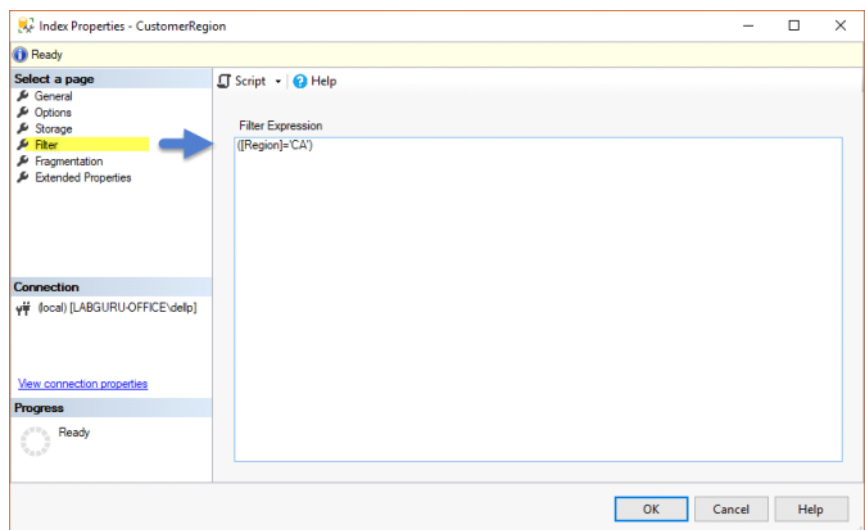
00b-FilteredIndex.s...ls (LABS\dellp (54)) X 00a-IndexInclude.sq...s (LABS\dellp

```
Use Northwind;
Go

SELECT Address, City, Region, PostalCode
FROM Customers
WHERE Region = 'CA';

CREATE INDEX CustomerRegion
ON Customers (Region)
Where Region='CA';

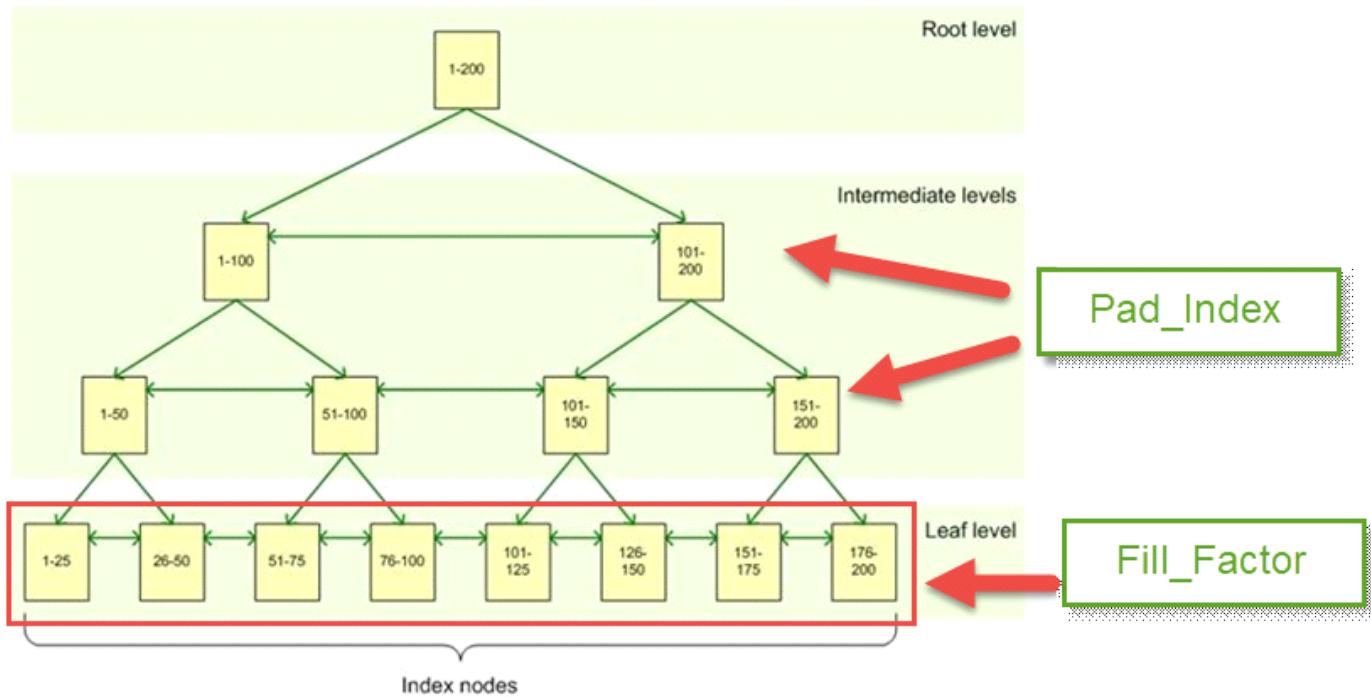
--Drop Index CustomerRegion On Customers
```



# Index Leaf Structure

Saturday, December 2, 2017 10:08 PM

[https://dbamohsin.wordpress.com/tag/pad\\_index/](https://dbamohsin.wordpress.com/tag/pad_index/)



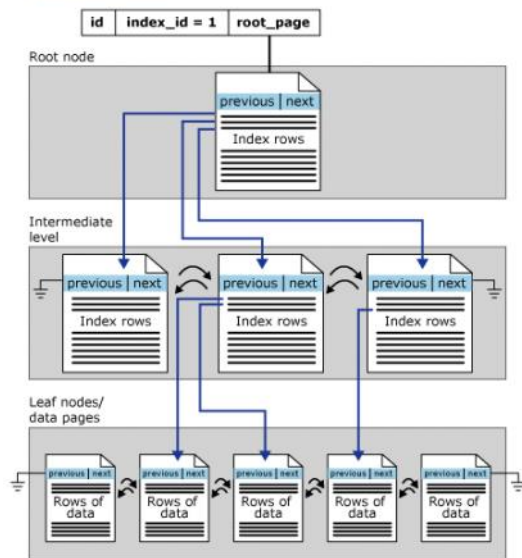
## Fill-Factor & Pad\_Index

Friday, February 8, 2019 9:56 PM

- **FILLFACTOR** applies to the bottom layer  
This is the leaf node/data layer in the picture below
- **PAD\_INDEX ON** means "Apply FILLFACTOR to all layers"  
This is the root node and intermediate level in the picture below

This means that **PAD\_INDEX** is only useful if **FILLFACTOR** is set. FILLFACTOR determines how much free space in an data page (roughly)

A picture from MSDN:



share improve this answer

answered Jul 28 '11 at 10:02



gbn

291k 45 414 512

On this page [msdn.microsoft.com/en-us/library/ms186869.aspx](https://msdn.microsoft.com/en-us/library/ms186869.aspx) it says when pad\_index is on, "The percentage of free space that is specified by FILLFACTOR is applied to the intermediate-level pages of the index". Will it also apply to the root level? Maybe it's just a books online oversight. – Ogrish Man Dec 4 '16 at 13:32