INDEXES IN SQL SERVER (II)

An **index** is a structure associated to a table or a view that optimize the access time to the records of the table or of the view.

Indexed views

- The first index created on a view must be a unique clustered index.
- After this one can be created the nonclustered indexes.

Creating a unique clustered index on a view *improves query performance* (the view is stored in the database in the same way a table with a clustered index is stored).
The query optimizer may use indexed views to speed up the query execution.

The steps required to create an indexed view (successful implementation of the indexed view):

1. Verify the SET options are correct for all existing tables that will be referenced in the view.
2. Verify that the SET options for the session are set correctly before you create any tables and the view.
3. Verify that the view definition is deterministic.
4. Create the view by using the WITH SCHEMABINDING option.
5. Create the unique clustered index on the view.

To maintain the views correctly and return consistent results, indexed views require fixed values for several SET options, when the conditions occur:

- The view and subsequent indexes on the view are created.
- The base tables referenced in the view at the time the table is created.
- There is any insert, update, or delete operation performed on any table that participates in the indexed view. Also, for operations like bulk copy, replication, and distributed queries.
- The indexed view is used by the query optimizer to produce the query plan.

SET options	**Required value**	**Default server value**	**Default OLE DB and ODBC value**	**Default DB-Library value**
ANSI_NULLS	ON	ON	ON	OFF
ANSI_PADDING	ON	ON	ON	OFF
ANSI_WARNINGS	ON	ON	ON	OFF
ARITHABORT	ON	ON	OFF	OFF
CONCAT_NULL_YIELDS_NULL	ON	ON	ON	OFF
NUMERIC_ROUNDABORT	OFF	OFF	OFF	OFF
QUOTED_IDENTIFIER	ON	ON	ON	OFF

Setting ANSI_WARNINGS to ON implicitly sets ARITHABORT to ON.

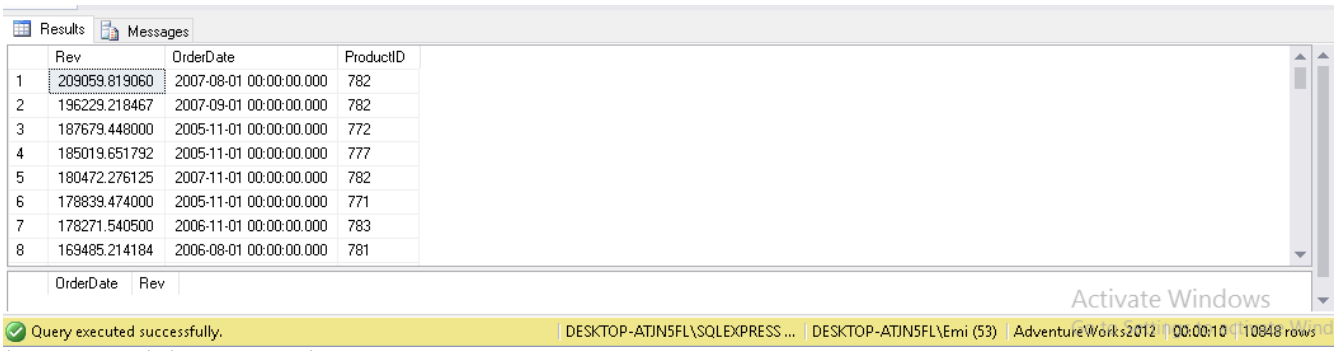
Example

```
-- INDEXED VIEWS
use AdventureWorks2012
go

--Set the options to support indexed views.
SET NUMERIC_ROUNDABORT OFF;
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, ARITHABORT,
```

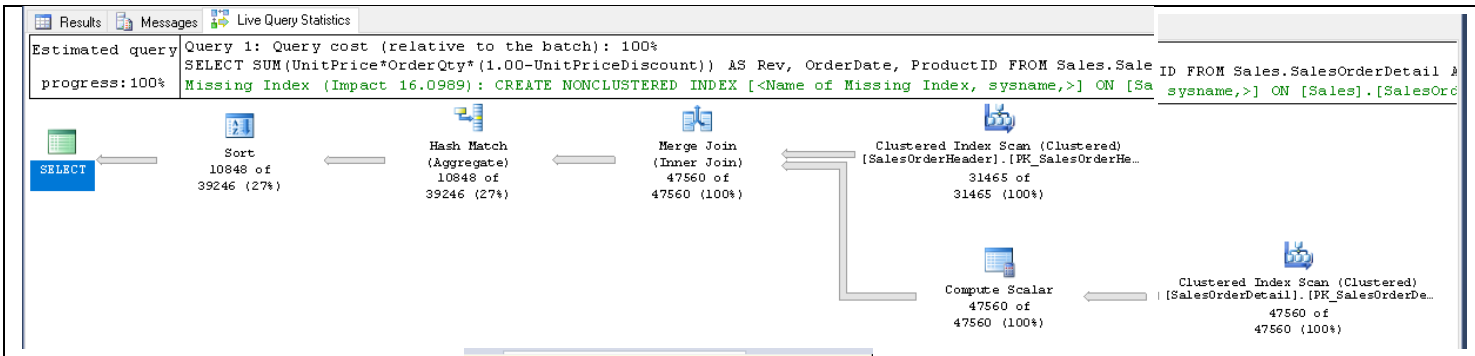
```
    QUOTED_IDENTIFIER, ANSI_NULLS ON;
GO
--Create view with schemabinding.
IF OBJECT_ID ('Sales.vOrders', 'view') IS NOT NULL
DROP VIEW Sales.vOrders ;
GO
CREATE VIEW Sales.vOrders
WITH SCHEMABINDING
AS
    SELECT SUM(UnitPrice*OrderQty*(1.00-UnitPriceDiscount)) AS Revenue,
        OrderDate, ProductID, COUNT_BIG(*) AS COUNT
    FROM Sales.SalesOrderDetail AS od, Sales.SalesOrderHeader AS o
    WHERE od.SalesOrderID = o.SalesOrderID
    GROUP BY OrderDate, ProductID;
GO
--Create an index on the view.
CREATE UNIQUE CLUSTERED INDEX IDX_V1
    ON Sales.vOrders (OrderDate, ProductID);
GO
--This query can use the indexed view even though the view is
--not specified in the FROM clause.
SELECT SUM(UnitPrice*OrderQty*(1.00-UnitPriceDiscount)) AS Rev,
    OrderDate, ProductID
FROM Sales.SalesOrderDetail AS od
    JOIN Sales.SalesOrderHeader AS o ON od.SalesOrderID=o.SalesOrderID
        AND ProductID BETWEEN 700 and 800
        AND OrderDate >= CONVERT(datetime,'05/01/2002',101)
GROUP BY OrderDate, ProductID
ORDER BY Rev DESC;
GO
--This query can use the above indexed view.
SELECT  OrderDate, SUM(UnitPrice*OrderQty*(1.00-UnitPriceDiscount)) AS Rev
FROM Sales.SalesOrderDetail AS od
    JOIN Sales.SalesOrderHeader AS o ON od.SalesOrderID=o.SalesOrderID
        AND DATEPART(mm,OrderDate)= 3
        AND DATEPART(yy,OrderDate) = 2002
GROUP BY OrderDate
ORDER BY OrderDate ASC;
GO

----------------------------------------
```




```
(10848 row(s) affected)

(0 row(s) affected)
```



Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	31465
Actual Number of Rows	31465
Actual Number of Batches	0
Estimated I/O Cost	0.509792
Estimated Operator Cost	0.54456 (9%)
Estimated Subtree Cost	0.54456
Estimated CPU Cost	0.0347685
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	31465
Estimated Row Size	19 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	3

Predicate

[AdventureWorks2012].[Sales].[SalesOrderHeader].[OrderDate] as [o].[OrderDate]>='2002-05-01 00:00:00.000'

Object

[AdventureWorks2012].[Sales].[SalesOrderHeader].[PK_SalesOrderHeader_SalesOrderID] [o]

Output List

[AdventureWorks2012].[Sales].[SalesOrderHeader].SalesOrderID, [AdventureWorks2012].[Sales].[SalesOrderHeader].OrderDate

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	121317
Actual Number of Rows	47560
Actual Number of Batches	0
Estimated I/O Cost	0.918681
Estimated Operator Cost	1.05229 (17%)
Estimated Subtree Cost	1.05229
Estimated CPU Cost	0.133606
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	47560
Estimated Row Size	37 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	5

Predicate

[AdventureWorks2012].[Sales].[SalesOrderDetail].[ProductID] as [od].[ProductID]>=(700) AND [AdventureWorks2012].[Sales].[SalesOrderDetail].[ProductID] as [od].[ProductID]<=(800)

Object

[AdventureWorks2012].[Sales].[SalesOrderDetail].[PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID] [od]

Output List

[AdventureWorks2012].[Sales].[SalesOrderDetail].SalesOrderID, [AdventureWorks2012].[Sales].[SalesOrderDetail].SalesOrderDetailID, [AdventureWorks2012].[Sales].[SalesOrderDetail].OrderQty, [AdventureWorks2012].[Sales].[SalesOrderDetail].ProductID, [AdventureWorks2012].[Sales].[SalesOrderDetail].UnitPrice, [AdventureWorks2012].[Sales].[SalesOrderDetail].UnitPriceDiscount


```
-- example 2
CREATE TABLE MyBigTable(
ItemID            INT PRIMARY KEY,
ItemDsc           VARCHAR(20),
QTY               INT)
GO

CREATE VIEW MyView WITH SCHEMABINDING AS
       SELECT ItemID, QTY
       FROM dbo.MyBigTable
       WHERE QTY > 10
GO

CREATE UNIQUE CLUSTERED INDEX idx_MyView ON MyView(QTY)
```

```
SELECT ItemID
FROM MyBigTable
WHERE QTY > 30
--
-- create table
-- create view with schemabiding
-- create unique clustered index on the table on a field used in a where clause
-- select from the table in which is involved the field/fields involved in the unique clustered index
from the view

SELECT ItemID
FROM MyView
WHERE QTY > 30
-- in both cases, the Unique Clustered Index is used :)
-- having indexed views means that you can create Unique Clustered Indexes on the fields that are not
involved in the primary key, and also, that these indexes are used in the queries created on the tables
or views.
```

Indexed views - Restrictions

• The SELECT queries cannot refer to other views.

• The SELECT queries must be deterministic.

A view is deterministic if all expressions in the select list, including the WHERE and GROUP BY clauses, are deterministic. Deterministic expressions always return the same result any time they are evaluated with a specific set of input values. Only deterministic functions can participate in deterministic expressions.

- DATEADD function is deterministic because it always returns the same result for any given set of argument values for its three parameters.
- GETDATE is not deterministic because it is always invoked with the same argument, but the value it returns changes each time it is executed.

To determine whether a view column is deterministic, use the **IsDeterministic** property of the COLUMNPROPERTY function. To determine if a deterministic column in a view with schema binding is precise, use the **IsPrecise** property of the COLUMNPROPERTY function. COLUMNPROPERTY returns 1 if TRUE, 0 if FALSE, and NULL for input that is not valid. This means the column is not deterministic or not precise.
Even if an expression is deterministic, if it contains float expressions, the exact result may depend on the processor architecture or version of microcode. To ensure data integrity, such expressions can participate only as non-key columns of indexed views. Deterministic expressions that do not contain float expressions are called precise. Only precise deterministic expressions can participate in key columns and in WHERE or GROUP BY clauses of indexed views.

• AVG, MIN, MAX, STDEV, STDEVP, VAR and VARP are not allowed.

• The index must be unique and *clustered*

• The SELECT queries cannot contain other queries inside, outer joins, EXCEPT, INTERSECT, TOP, UNION, ORDER BY, DISTINCT and so on.

The following types of queries can achieve significant performance benefits if a view that is referenced by the corresponding query is indexed:

- Queries that process many rows and contain join operations or aggregate functions

- Join operations and aggregate functions that are frequently performed by one or several queries

Do	Not Do
- The view definition can reference one or more tables in the same database. - Once the unique clustered index is created, additional nonclustered indexes can be created against the view. - You can update the data in the underlying tables – including inserts, updates, deletes, and even truncates.	- The view definition can't reference other views, or tables in other databases. - It can't contain COUNT, MIN, MAX, TOP, outer joins, or a few other keywords or elements. - You can't modify the underlying tables and columns. The view is created with the WITH SCHEMABINDING option. - You can't always predict what the query optimizer will do.

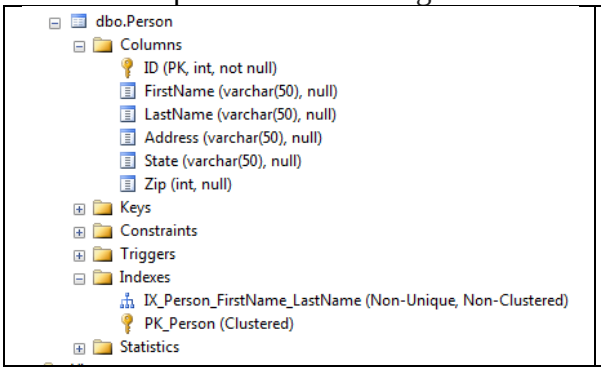
Indexing – Rules and Good Practices

- Each table must have a clustered index; ideally this one should have reduced dimensions, to be selective, growing and static (a table without clustered indexed is called *heap*).
- For the foreign keys can be created nonclustered indexes.
- For the fields used in the WHERE clause can be created nonclustered indexes.
- We won't have simple indexes – with one column – on each field of a table; these ones will complicate the maintenance of the table.
- In the composed indexes, the most selective field ("the closer" to unique) will be the first from the key.
- For the most of the queries can be created nonclustered covering indexes.

Fragmentation

- **Internal fragmentation**: the records are not keep in a continuous part from the interior of the page. The internal fragmentation appear if in a page there is space not used between records. The fulfillment degree of a page can vary in time. The space that is not used can take to an inefficient use of the cache and to more page transfers between the disk memory and the internal memory, fact that in the end affect the performance of the queries. (It is caused by pages that have too much free space. Let's pretend at the beginning of the day we have a table with 40 pages that are 100% full, but by the end of the day we have a table with 50 pages that are only 80% full because of various delete and insert statements throughout the day. This causes an issue because now when we need to read from this table we have to scan 50 pages instead of 40 which should may result in a decrease in performance.)

Example for Internal Fragmentation:

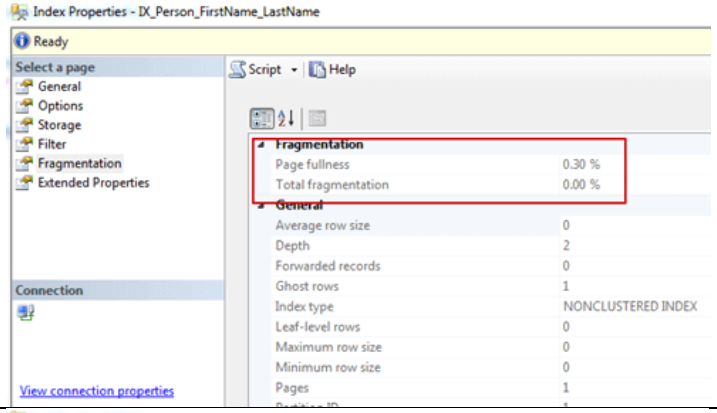
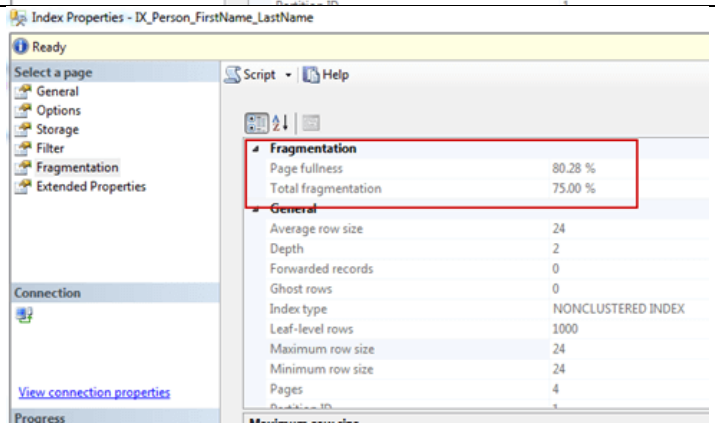



```
-- drop index N_idx_Person_FirstName_LastName ON Person
-- drop table Person

create table Person(
ID int primary key identity(1,1),
FirstName varchar(50),
LastName varchar(50),
Address varchar(50), -- refer the street
State varchar(50),
zip int
)
```

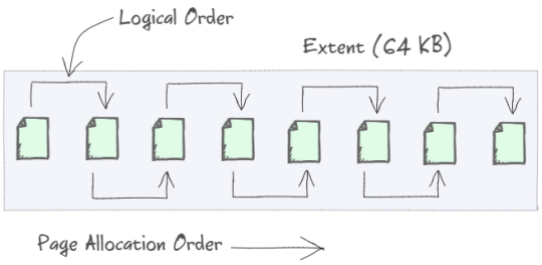
<table>
<tr><td></td><td>

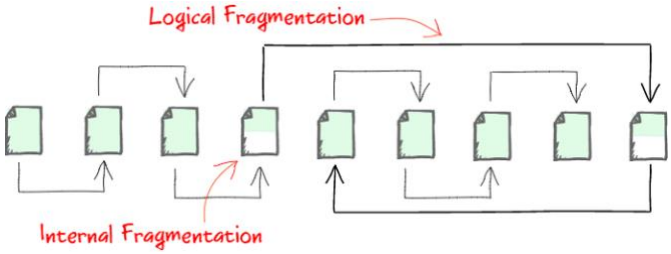
```
create nonclustered index N_idx_Person_FirstName_LastName
on Person(FirstName, LastName)
```

</td></tr>
<tr><td>Right click on the index, click Properties, and Fragmentation to see fragmentation and page fullness. This is a brand new index so it's at 0% fragmentation.</td><td> 
</td></tr>
<tr><td>INSERT INTO Person VALUES (‘Brady’, ‘Upon’, ‘123 Main Street’, ‘TN’ 55555)
GO 1000</td><td> 

The index becomes 75% fragmented and the average percent of full pages (page fullness) increases to 80%.</td></tr>
<tr><td colspan="2">So, the performance will be affected by the size of the table and by the page counts.</td></tr>
</table>

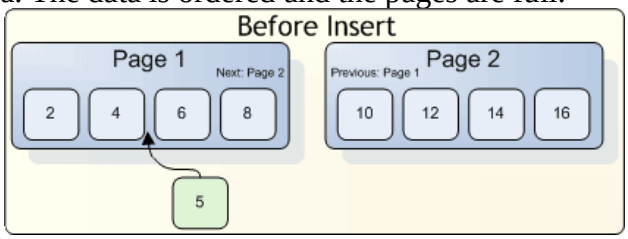
- **Extent fragmentation**: on the disk, pages and extensions (extension – group of 8 memory pages) – are not stored in a continuous space; when the extensions of a table are not stored continuously on the disc, the passing from an extension to another can cause bigger rotations of the disk. (It is caused by pages that are out of order. At the beginning of the day we have a perfectly ordered table. During the day we issue hundreds of update statements possibly leaving some empty space on one page and trying to fit space into other pages. This means our storage has to jump around to obtain the data needed instead of reading in one direction.)
- **Logical fragmentation**: Each page of an index is linked to the previous one and the following one in the logic order of the key values. Because some of the pages become full and also because of the value redistribution (*Page Split*), the pages become out-of-order.
    - O page out-of-order is a page for which the next physic page from the index is not the next logical page.



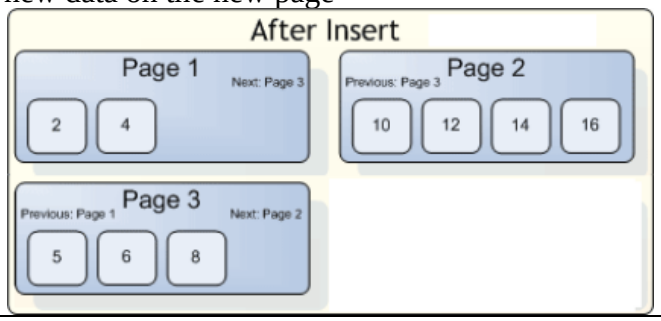


Example 1 for logical fragmentation:
Imagine there are two data pages for a table with a clustered index.

a. The data is ordered and the pages are full.

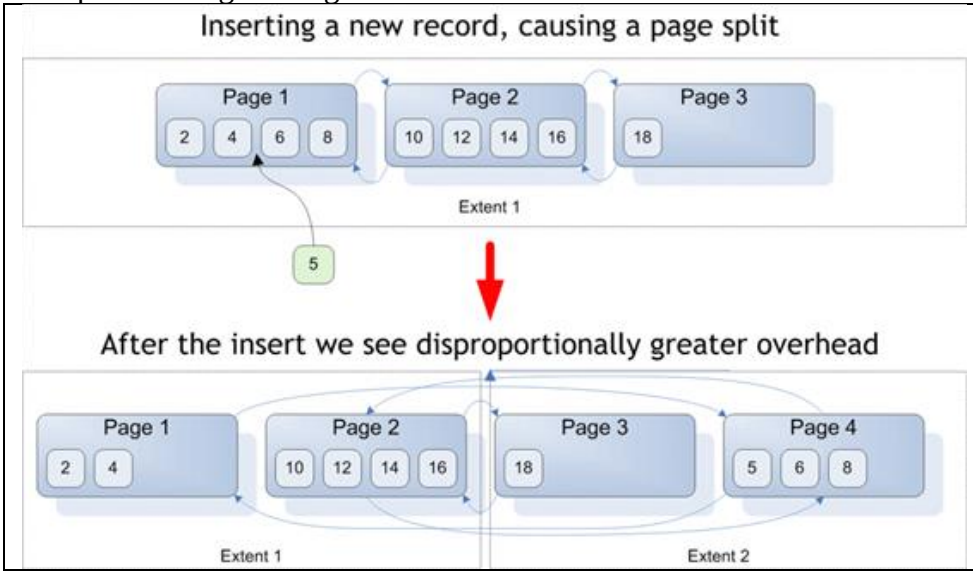


b. A new row with a primary key of "5" needs to be inserted, and since it is a clustered index, the new row is inserted in order. Because the target page is full enough that the new row does not fit, SQL Server splits the page roughly in half and inserts the new data on the new page

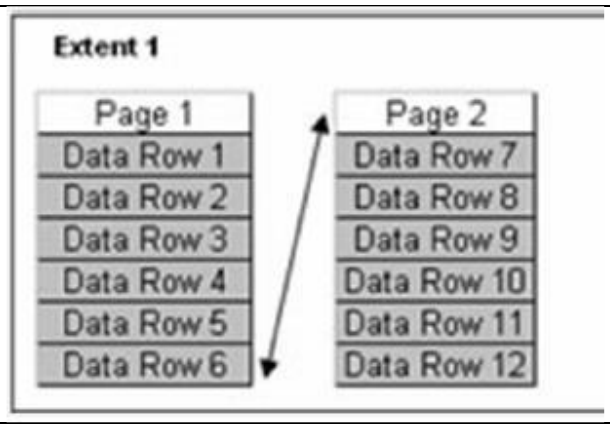


c. Now, the logical order of the index does not match the physical order, and the index has become fragmented.

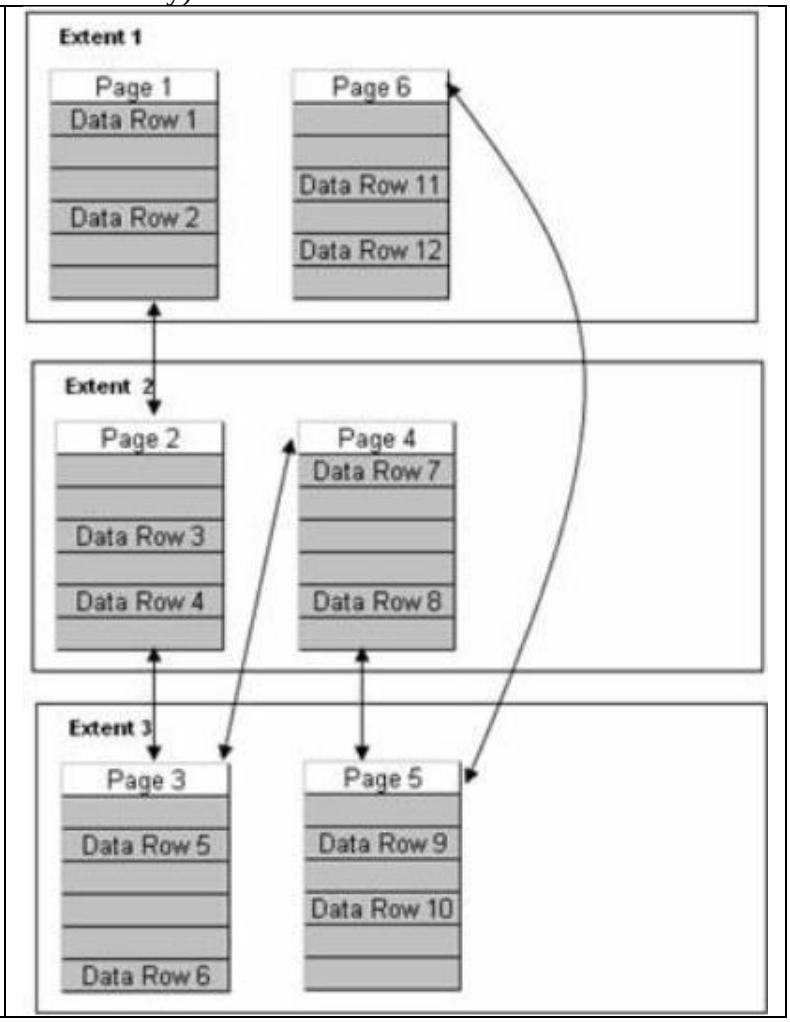
Example 2 for logical fragmentation:



Example for no fragmentation (on the beginning of the day):

Request to read pages: 2 Changing extensions: 0 Space on the disk used by the table: 16 KB avg_fragmentation_in_percent: 0 avg_page_space_used_in_percent: 100 (the pages are 'full' – no internal fragmentation – just follows one to another)	

Example for all the fragmentations (on the end of the day):

Request to read pages: 6 Changing extensions: 5 Space on the disk used by the table: 48 KB avg_fragmentation_in_percent> 80 avg_page_space_used_in_percent: 33 (- internal fragmentation (because the pages are NOT 'full') and - extent fragmentation (because the pages are not stored continuous on the disk) and - logical fragmentation (because we 'jump' between pages))	

To analyze the fragmentation, we can use the dynamic view (DMV) ***sys.dm_db_index_physical_stats*** (that returns the size and fragmentation information for the data and indexes of the specified table or view), with

•avg_fragmentation_in_percent

•percentage value -

- heaps–extent fragmentation
- indexes–logical fragmentation

- avg_page_space_used_in_percent
  - average percentage of available space in all pages

```
-- detect fragmentation
SELECT I.index_id, I.name,
  DM.avg_fragmentation_in_percent,
  DM.avg_page_space_used_in_percent
FROM sys.dm_db_index_physical_stats(db_id(),  NULL, NULL, NULL, DEFAULT) DM
JOIN sys.Indexes I ON I.object_id=DM.object_id AND I.Index_id=DM.index_id
```

Results | Messages

	index_id	name	avg_fragmentation_in_percent	avg_page_space_used_in_percent
24	1	PK_SpecialOffer_SpecialOfferID	0	NULL
25	2	AK_SpecialOffer_rowguid	0	NULL
26	1	PK_ErrorLog_ErrorLogID	0	NULL
27	1	PK_ProductListPriceHistory_ProductID...	0	NULL
28	1	PK_Address_AddressID	2.03488372093023	NULL
29	1	PK_Address_AddressID	0	NULL
30	1	PK_Address_AddressID	0	NULL
31	2	AK_Address_rowguid	0	NULL
32	3	IX_Address_AddressLine1_AddressLin...	0	NULL

```
SELECT OBJECT_NAME(ips.OBJECT_ID), i.NAME, ips.index_id, index_type_desc,
avg_fragmentation_in_percent, avg_page_space_used_in_percent, page_count
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'SAMPLED') ips
INNER JOIN sys.indexes i ON (ips.object_id = i.object_id) AND (ips.index_id = i.index_id)
ORDER BY avg_fragmentation_in_percent DESC
```

Results | Messages

	(No column name)	NAME	index_id	index_type_desc	avg_fragmentation_in_percent	avg_page_space_used_in_percent	page_count
1	SpecialOfferProduct	AK_SpecialOfferProduct_rowguid	2	NONCLUSTERED INDEX	50	99.6787744007907	2
2	StateProvince	PK_StateProvince_StateProvinceID	1	CLUSTERED INDEX	50	85.9154929577465	2
3	Store	IX_Store_SalesPersonID	3	NONCLUSTERED INDEX	50	60.6004447739066	2
4	ProductProductPhoto	PK_ProductProductPhoto_ProductID_ProductPhotoID	2	NONCLUSTERED INDEX	50	68.4704719545342	2
5	ProductReview	PK_ProductReview_ProductReviewID	1	CLUSTERED INDEX	50	63.9918458117124	2
6	ProductVendor	IX_ProductVendor_UnitMeasureCode	2	NONCLUSTERED INDEX	50	56.8075117370892	2
7	Vendor	PK_Vendor_BusinessEntityID	1	CLUSTERED INDEX	50	58.3271559179639	2

Query executed successfully. | DESKTOP-ATJN5FL\SQLEXPRESS ... | DESKTOP-ATJN5FL\Emi (53) | AdventureWorks2012 | 00:00:14 | 241 rows

Results | Messages

	(No column name)	name	index_id	index_type_desc	avg_fragmentation_in_percent	avg_page_space_used_in_percent	page_count
1	RPCust	IX_RPCust_1	3	NONCLUSTERED INDEX	98.8296488946684	55.8104274771436	769
2	tblTripBasePrice	IX_tblTripBasePrice_1	3	NONCLUSTERED INDEX	97.2727272727273	58.4332962688411	110
3	tblTripLog	VehicleID	8	NONCLUSTERED INDEX	97.2222222222222	65.0543612552508	36
4	tblEmployeeDocumentArchive	{BE348B3E-935C-473A-8FC3-...	2	NONCLUSTERED INDEX	97.1428571428571	60.7822461082283	35
5	tblEmployeeDocumentArchive	DocumentTypeID	4	NONCLUSTERED INDEX	96.969696969697	64.4675067951569	33
6	tblESig	aaaaatblESig_PK	5	NONCLUSTERED INDEX	96.551724137931	57.6698047936743	29

```
-- returns size and fragmentation statistics for all indexes and partitions of the
Person.Address table.
-- The scan mode is set to 'LIMITED' for best performance and to limit the statistics that are
returned
DECLARE @db_id SMALLINT;
DECLARE @object_id INT;

SET @db_id = DB_ID(N'AdventureWorks2012');
SET @object_id = OBJECT_ID(N'AdventureWorks2012.Person.Address');

IF @db_id IS NULL
BEGIN
    PRINT N'Invalid database';
END
ELSE IF @object_id IS NULL
BEGIN
    PRINT N'Invalid object';
END
ELSE
BEGIN
    SELECT * FROM sys.dm_db_index_physical_stats(@db_id, @object_id, NULL, NULL , 'LIMITED');
END
GO
```

Results | Messages

	database_id	object_id	index_id	partition_number	index_type_desc	alloc_unit_type_desc	index_depth	index_level	avg_fragmentation_in_percent	fragment_count	avg_fr
1	7	373576369	1	1	CLUSTERED INDEX	IN_ROW_DATA	2	0	2.0348837209302	16	21.5
2	7	373576369	1	1	CLUSTERED INDEX	ROW_OVERFLOW_DATA	1	0	0	NULL	NULL
3	7	373576369	1	1	CLUSTERED INDEX	LOB_DATA	1	0	0	NULL	NULL
4	7	373576369	2	1	NONCLUSTERED INDEX	IN_ROW_DATA	2	0	0	2	32
5	7	373576369	3	1	NONCLUSTERED INDEX	IN_ROW_DATA	3	0	0	9	23.444
6	7	373576369	4	1	NONCLUSTERED INDEX	IN_ROW_DATA	2	0	0	4	8.5
7	7	373576369	6	1	NONCLUSTERED INDEX	IN_ROW_DATA	2	0	0	2	105.5

```
-- returns all statistics for the heap dbo.DatabaseLog in the AdventureWorks2012 database
DECLARE @db_id SMALLINT;
DECLARE @object_id INT;
SET @db_id = DB_ID(N'AdventureWorks2012');
SET @object_id = OBJECT_ID(N'AdventureWorks2012.dbo.DatabaseLog');
IF @object_id IS NULL
BEGIN
    PRINT N'Invalid object';
END
ELSE
BEGIN
    SELECT * FROM sys.dm_db_index_physical_stats(@db_id, @object_id, 0, NULL , 'DETAILED');
END
GO
```

Results | Messages

	database_id	object_id	index_id	partition_number	index_type_desc	alloc_unit_type_desc	index_depth	index_level	avg_fragmentation_in_percent	fragment_count	avg_fragment_size_i
1	7	245575913	0	1	HEAP	IN_ROW_DATA	1	0	31.3131313131313	36	21.7222222222222
2	7	245575913	0	1	HEAP	LOB_DATA	1	0	0	NULL	NULL

```
-- returns all statistics for all tables and indexes within the instance of SQL Server by
specifying the wildcard NULL for all parameters.
SELECT * FROM sys.dm_db_index_physical_stats (NULL, NULL, NULL, NULL, NULL);
GO
```

Results | Messages

	database_id	object_id	index_id	partition_number	index_type_desc	alloc_unit_type_desc	index_depth	index_level	avg_fragmentation_in_percent	fragment_count	avg_fragment_
1	1	7671075	0	1	HEAP	IN_ROW_DATA	0	0	0	0	0
2	1	39671189	0	1	HEAP	IN_ROW_DATA	0	0	0	0	0
3	1	52195236	1	1	CLUSTERED INDEX	IN_ROW_DATA	0	0	0	0	0
4	1	117575457	0	1	HEAP	IN_ROW_DATA	0	0	0	0	0
5	1	133575514	0	1	HEAP	IN_ROW_DATA	0	0	0	0	0

Also, to analyze the fragmentation, can be used the ***DBCC SHOWCONTIG***

```
2
3  DBCC SHOWCONTIG
```

100 %

Messages

```
DBCC SHOWCONTIG scanning 'tblAgentLogin' table...
Table: 'tblAgentLogin' (1487134); index ID: 0, database ID: 7
TABLE level scan performed.
- Pages Scanned................................: 1
- Extents Scanned..............................: 1
- Extent Switches..............................: 0
- Avg. Pages per Extent........................: 1.0
- Scan Density [Best Count:Actual Count].......: 100.00% [1:1]
- Extent Scan Fragmentation ...................: 0.00%
- Avg. Bytes Free per Page.....................: 8057.0
- Avg. Page Density (full).....................: 0.46%
DBCC SHOWCONTIG scanning 'tblRentalTripNameExclusion' table...
Table: 'tblRentalTripNameExclusion' (3491787); index ID: 1, database ID: 7
TABLE level scan performed.
- Pages Scanned................................: 0
- Extents Scanned..............................: 0
- Extent Switches..............................: 0
- Avg. Pages per Extent........................: 0.0
- Scan Density [Best Count:Actual Count].......: 100.00% [0:0]
- Logical Scan Fragmentation ..................: 0.00%
- Extent Scan Fragmentation ...................: 0.00%
- Avg. Bytes Free per Page.....................: 0.0
- Avg. Page Density (full).....................: 0.00%
```

To reduce fragmentation (Defragmentation):

a. In a HEAP - create a clustered index on the table, then drop it

By creating the clustered index, the records are rearranged in an order, and then are placed in a contiguous area on the disk.

b. In an index - use the [ALTER INDEX] [REBUILD | REORGANIZE] command, in the following way: take the value of the *avg_fragmentation_in_percent* from *sys.dm_db_index_physical_stats*, and decide the case:

0 to 5-10% — do nothing
5-10% to 30% — do REORGANIZE
30% to 100% — do REBUILD

1. Alter index reorganize

- Rearranges only leaf pages, and compresses index pages that delete empty pages.
- Less effective than rebuilding indexes.

- When 5%<=avg_fragmentation_in_percent<=30% and the count of pages higher than 2000
- Warning : Does not update statistics.

```
USE AdventureWorks2012;
GO
-- Reorganize all indexes on the HumanResources.Employee table.
ALTER INDEX ALL ON HumanResources.Employee
REORGANIZE ;
GO
Command(s) completed successfully.
```

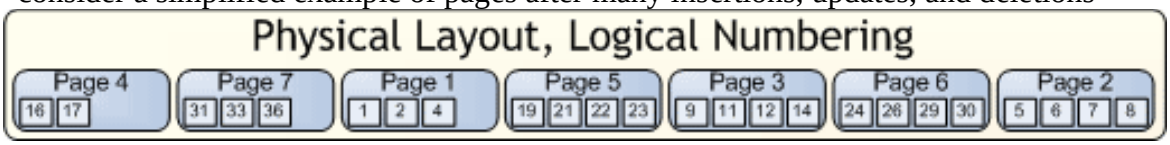
2. Alter index rebuild

- Recreating the index, when the index is clustered, table is also reorganized.
- The reconstruction of a clustered index with [ALTER INDEX REBUILD] does not rebuild nonclustered indexes on the table, unless [ALL] is specified.
- When avg_fragmentation_in_percent>30% and the count of pages higher than 2000
- The [ONLINE=ON] option allows you to rebuild an index without blocking the activity.

```
USE AdventureWorks2012;
GO
ALTER INDEX PK_Employee_BusinessEntityID ON HumanResources.Employee
REBUILD;
GO
Command(s) completed successfully.
```

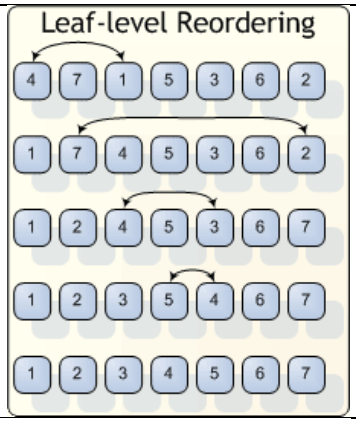
When drop and recreate a clustered index?
- when a clustered index is created, the data is redistributed -> full data pages
- the level of fullness can be configured with the FILLFACTOR option in CREATE INDEX

Example of defragmentation of an index (with Reorganize)
- consider a simplified example of pages after many insertions, updates, and deletions



Page numbering represents the logical sequence of pages. The physical sequence in the figure from left to right does not correspond to the logical sequence.

During the first pass, SQL Server finds the first physical page (4) and the first logical page (1), and then exchanges these pages in a discrete transaction.
On the second pass, SQL Server exchanges the next physical page (7) with next logical page (2).
On the third pass, SQL Server exchanges the next physical page (4) with the next logical page (3).
On the fourth pass, SQL Server exchanges the next physical page (5) with the next logical page (4).



T-SQL -Control-of-Flow Language

•BEGIN...END
•RETURN
•BREAK
•THROW
•CONTINUE
•TRY...CATCH
•GOTO label
•WAITFOR
•IF...ELSE
•WHILE

RETURN

RETURN [integer_expression]

• exits from a procedure / batch / statement block
• returning status codes
• unless specified otherwise, system stored procedures return:
• 0 - success
• a non zero value – failure

```
-- return
CREATE PROCEDURE uspCheckCountry @country
varchar(50)
AS
IF @country = 'Romania'
RETURN 1
ELSE
RETURN 2;
GO
```

```
DECLARE @ret_status_code int
EXEC @ret_status_code= uspCheckCountry 'Romania'
SELECT @ret_status_code
GO
-- 1
```

```
DECLARE @ret_status_code int
EXEC @ret_status_code= uspCheckCountry 'Spain'
SELECT @ret_status_code
GO
-- 2
```

WHILE

WHILE boolean_expression {sql_statement / statement_block / BREAK / CONTINUE}

- repeated execution of a SQL statement or statement block while the specifiec condition is true
- BREAK
  - exists current WHILE loop (if the latter is nested inside another WHILE loop, BREAK exists only the current loop)
  - can appear in an IF statement
- CONTINUE
  - restarts a WHILE loop
  - any statements after CONTINUE are ignored

```
-- with break
DECLARE @i INT = 1;
WHILE @i <= 5
BEGIN
    SET @i = @i + 1;
    IF @i = 4
        BREAK;
    PRINT @i;
```

```
2
3
```

END	
-- with break & Continue DECLARE @i INT = 1; WHILE @i <= 8 BEGIN SET @i = @i + 1; IF @i = 4 CONTINUE; IF @i = 6 BREAK; PRINT @i; END	2 3 5

GOTO

Execution continues at the label

GOTO label

```
DECLARE @i int = 1;
WHILE @i <=8
BEGIN
    SET @i = @i + 1
    IF @i = 4 GOTO Label1 -- goes to Label1
    IF @i = 6 GOTO Label2  -- goes to Label2
END
Label1:
    SELECT 'First Label'
Label2:
    PRINT 'Secund Label';
```

Results | Messages

	(No column name)
1	First Label

```
(1 row(s) affected)
Secund Label
```

WAITFOR

WAITFOR {DELAY 'time_to_pass' / TIME 'time_to_execute'}

- blocks the execution of a batch / stored procedure / transaction

```
-- execution continues at 07:15
WAITFOR TIME '07:15'
-- execution continues after 3 hours
WAITFOR DELAY '03:00'
```

- if the server is busy, the counter may not start immediately -> the delay may be longer than the specified one.

```
select Top 1 * from Person
waitfor delay '00:00:04'
select Top 1 * from Person
```

Results | Messages

	ID	FirstName	LastName	Address	State	zip
1	1	Brady	Upon	123 Main Street	TN	55555

	ID	FirstName	LastName	Address	State	zip
1	1	Brady	Upon	123 Main Street	TN	55555

Query executed su... | DESKTOP-ATJN5FL\SQLEXPRESS ... | DESKTOP-ATJN5FL\Emi (55) | AdventureWorks2012 | 00:00:04 | 2 rows

THROW

THROW [
{error_number / @local_variable},
{message / @local_variable},
{state / @local_variable}] [;]

- raises an execption, transfers execution to the CATCH block of a TRY … CATCH construct
- exception severity – always 16

```
THROW 51000,'50 rows have been modified',1;
```

```
Msg 51000, Level 16, State 1, Line 58
50 rows have been modified
```

TRY … CATCH

BEGIN TRY
{sql_statement / statemen_block}
END TRY
BEGIN CATCH
[{sql_statement / statement_block}]
END CATCH [;]

- implements error handling in Transact-SQL
- catches execution errors with severity>10 and that do not close the database connection

•ERROR_NUMBER() – returns the error number

•ERROR_SEVERITY() – returns the error severity

•ERROR_STATE() – returns the error state number

•ERROR_PROCEDURE() – returns the name of the stored procedure / trigger where the error occurred

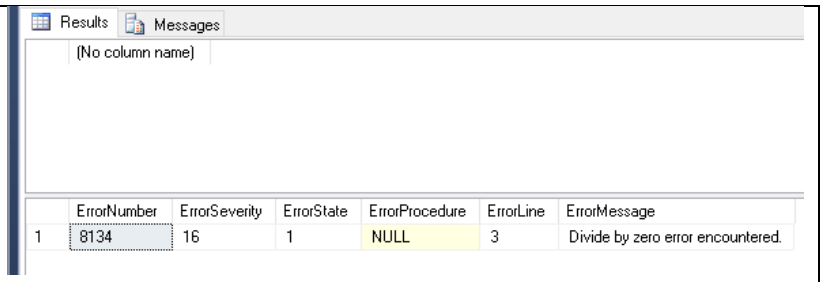
•ERROR_LINE() – returns the line number of the error occurrence

•ERROR_MESSAGE() – returns the error message

ERRORS

- error number
  - integer value between 1 and 49999
  - custom error messages: 50001…
- error severity
  - 26 severity levels
  - severity level ≥16 => error automatically logged
  - severity level between 20 and 25 => fatal error, the connection is terminated
- error message
  - up to 255 characters

```
-- Divide by zero error encountered.
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
              ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```




```
-- error not catched
BEGIN TRY
    -- Table does not exist; object name
resolution - error not caught.
    SELECT * FROM NonexistentTable;
END TRY
BEGIN CATCH
    SELECT
       ERROR_NUMBER() AS ErrorNumber,
       ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

```
Msg 208, Level 16, State 1, Line 113
Invalid object name 'NonexistentTable'.
```

References:

https://www.databasejournal.com/features/mssql/article.php/3867651/SQL-Server-Indexed-Views.htm
https://docs.microsoft.com/en-us/sql/relational-databases/views/create-indexed-views?view=sql-server-2017
https://www.brentozar.com/archive/2013/11/what-you-can-and-cant-do-with-indexed-views/
https://www.red-gate.com/simple-talk/sql/learn-sql-server/sql-server-indexed-views-the-basics/
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-2017
https://www.mssqltips.com/sqlservertip/4331/sql-server-index-fragmentation-overview/
https://www.idera.com/productssolutions/sqlserver/sqldefragmanager/what-is-fragmentation
https://social.technet.microsoft.com/wiki/contents/articles/40339.what-is-fragmentation-in-sql-server.aspx
https://logicalread.com/fix-sql-server-index-fragmentation-mc11/#.XCywRFwzbDc
https://blog.devart.com/sql-server-index-fragmentation-in-depth.html
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-index-physical-stats-transact-sql?view=sql-server-2017
https://www.sqlskills.com/blogs/paul/inside-sys-dm_db_index_physical_stats/
https://www.sqlservercentral.com/Forums/1086081/sysdmdbindexphysicalstats-Query
https://www.red-gate.com/simple-talk/blogs/quick-look-at-dm_db_index_physical_stats/
https://www.sqlservertutorial.net/sql-server-stored-procedures/sql-server-try-catch/
https://docs.microsoft.com/en-us/sql/t-sql/language-elements/try-catch-transact-sql?view=sql-server-ver15