Indexes

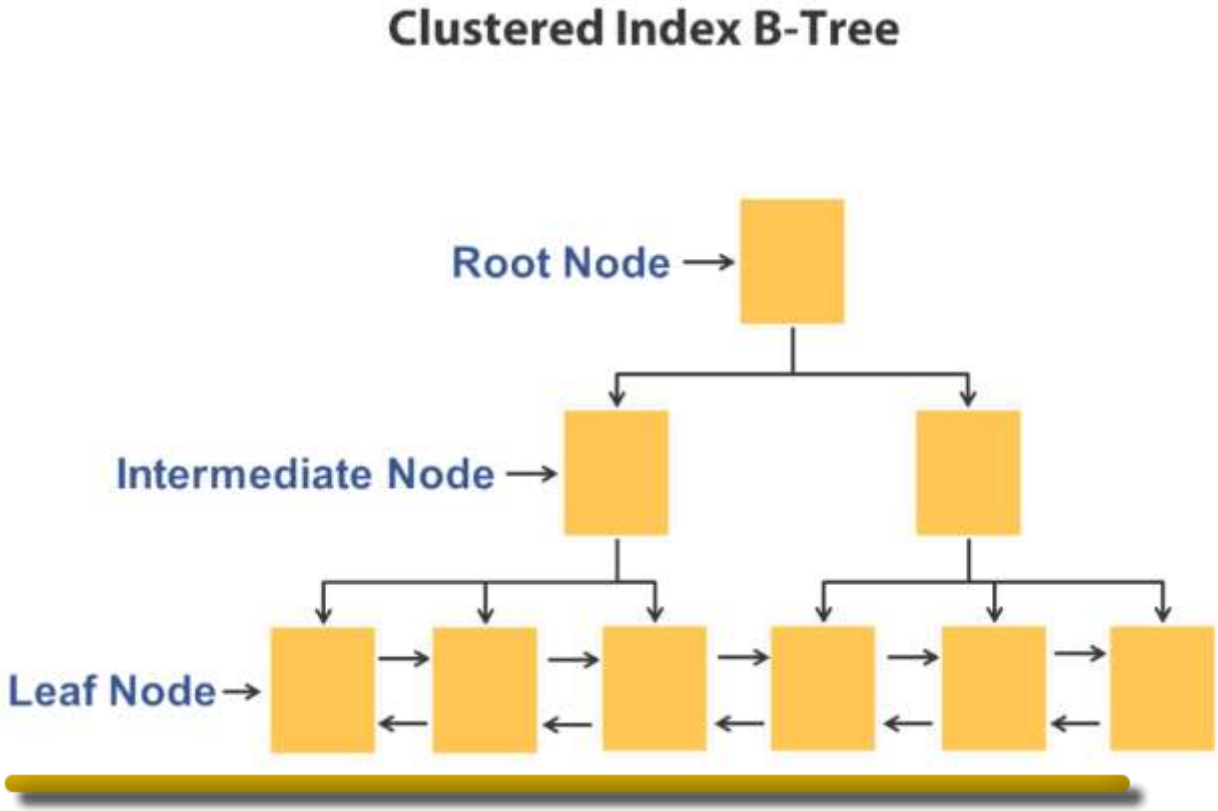
- What are they?
- Types of indexes
- Terms
  - Pages
  - Extents
  - Heap
  - Clustered Indexes
  - Non-Clustered indexes
  - Fill-Factor

Pages and Extents

- Page(8k)
  - is the fundamental unit of storage in SQL Server
  - So SQL Server databases have 128 pages per megabyte
  - Each page begins with a 96byte header
  - Max amount of data in a single row on a page is 8060 bytes
- Entents
  - A collection of 8 physical contiguous pages
- Heap
  - A table without a clustered index
  - The data rows are not stored in any particular order
  - Data pages are not linked in a linked list

Clustered Index

- The data rows are stored in order based on the clustered index key
- The clustered index is implemented as a B-Tree index structure
- Data pages in the leaf level are linked in a doubly-linked list
- Clustered indexes have one row in sys.partitions with index_id=1
    - =0 means have a heap
    - 1=clustered index
    - >1=have non clustered index

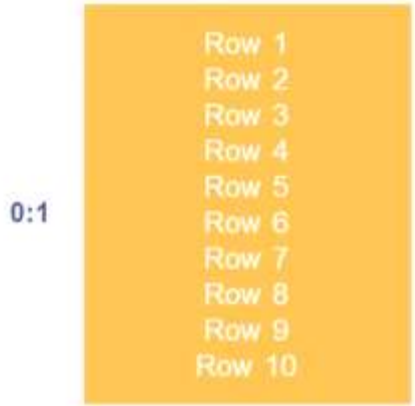


Non-clustered Index

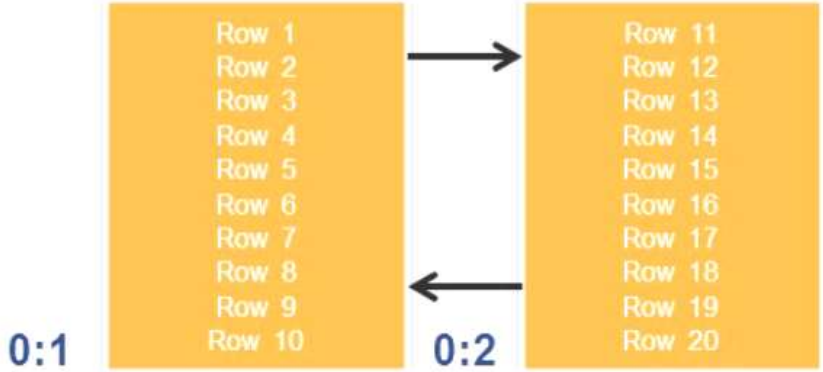
- Non-clustered indexes has a B-tree index structure like the one in a clustered index
    - Non-clustered indexes do not affect the order of the data rows
- Each index row contains the non-clustered key values, a row locator and any included, or nonkey, columns

Building B-Tree

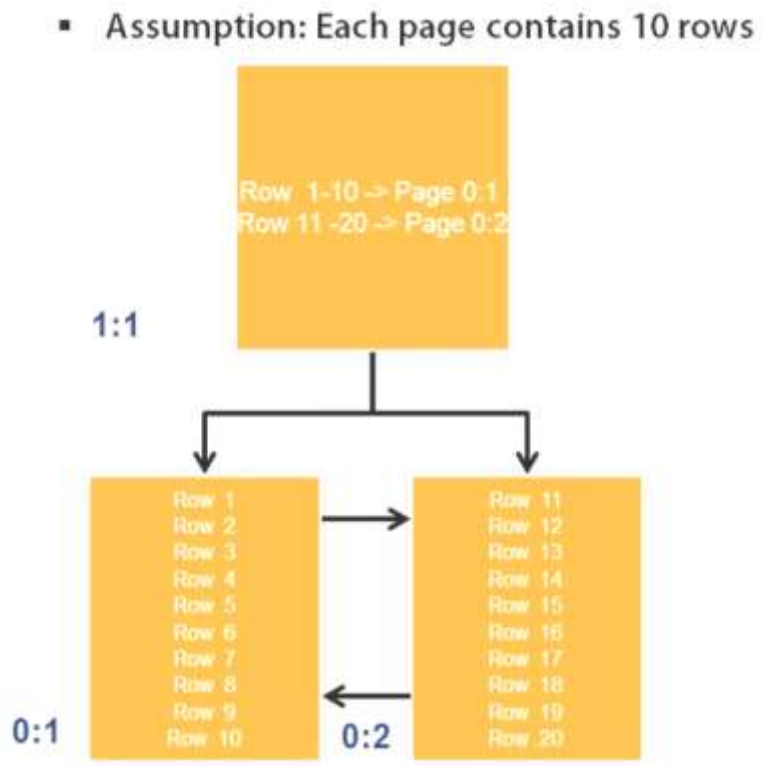
- Assumption: Each page contains 10 rows



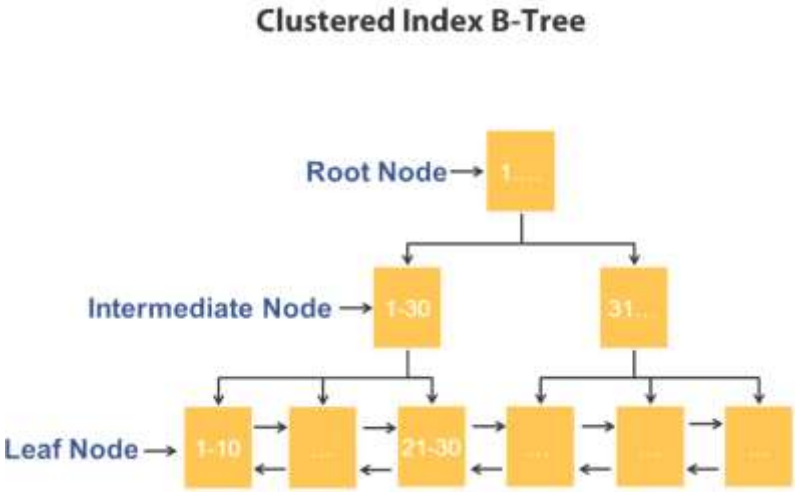
Add 10 more rows, it adds another page



Now you need page to manage it



Search for row3
Finds root node
Searches root node for 3



Goes thru intermediate nodes
Goes to 1-30
Goes to leaf of 3

Fill Factor

- The method of pre-allocating some space for future expansion
- To avoid PAGESPLITS and degrade performance

It does not preallocate unless you are using fixed data types
So a small row can become bigger in future

Demo
In SQL Server

SQL

```
Use tempDB
go

create table Indexing(
     ID int identity(1,1),
     Name char(4000),
     Company char(4000),
     Pay int)
```

RESULT

```
Messages
Command(s) completed successfully.
```

Create indexing table
So each row gets single page

SQL

```
Select
     OBJECT_NAME(object_id) TableName,
     ISNULL(name,OBJECT_NAME(object_id)) IndexName,
     Index_id,
     type_desc
     from sys.indexes
     where OBJECT_NAME(object_id)='Indexing'
```

RESULT

Results | Messages

	TableName	IndexName	Index_id	type_desc
1	Indexing	Indexing	0	HEAP

Query... | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | AdventureWorks2012 | 00:00:00 | 1 rows

Shows tablename indexing has type of index_id=0 and is a heap
Write this SQL to add 10000 items

SQL

```
SET NOCOUNT ON
--It suppresses the "xx rows affected" message
```

```
--after any DML
INSERT INTO Indexing VALUES
 ('test one', 'Expert',10000)
```

RESULT

Messages

```
Command(s) completed successfully.
```

100 %

Query... | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | AdventureWorks2012 | 00:00:00 | 0 rows

Add this status query

SQL

```
SELECT
     OBJECT_NAME(object_id)Name,
     index_type_desc as INDEXTYPE,
     index_id as INDEX_ID,
     index_depth as DEPTH,
     index_level as IND_LEVEL,
     record_count as RECORDCOUNT,
     page_count as PageCount,
     fragment_count as Fragmentation
     FROM sys.dm_db_index_physical_stats(DB_ID(),
OBJECT_ID('Indexing'), NULL, NULL, 'DETAILED')
GO
```

RESULT

Results | Messages

	Name	INDEXTYPE	INDEX_ID	DEPTH	IND_LEVEL	RECORDCOUNT	PageCount	Fragmentation
1	Indexing	HEAP	0	1	0	1	2	2

Query... | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | AdventureWorks2012 | 00:00:00 | 1 rows

- Set nocount to on
- To suppress endrows and is best practice
- Add 2 more records

SQL

```
     SET NOCOUNT ON
     --It suppresses the "xx rows affected" message
     --after any DML
     INSERT INTO Indexing VALUES
     ('Steve', 'Central',15000)
```

```
      ,('Joe', 'SQLAuthority', 13000)
      Go
```

Select status check

SQL

```
SELECT
      OBJECT_NAME(object_id)Name,
      index_type_desc as INDEXTYPE,
      index_id as INDEX_ID,
      index_depth as DEPTH,
      index_level as IND_LEVEL,
      record_count as RECORDCOUNT,
      page_count as PageCount,
      fragment_count as Fragmentation
      FROM sys.dm_db_index_physical_stats(DB_ID(),
OBJECT_ID('Indexing'), NULL, NULL, 'DETAILED')
GO
```

RESULT

Results

	Name	INDEXTYPE	INDEX_ID	DEPTH	IND_LEVEL	RECORDCOUNT	PageCount	Fragmentation
1	Indexing	HEAP	0	1	0	3	3	2

Query... | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | AdventureWorks2012 | 00:00:00 | 1 rows

Get page counts , fragmentation etc

Insert 100 records

SQL

```
insert into indexing values
 ('Dummy', 'Dummy Company', 1000)
GO 100
```

RESULT

Messages

```
Beginning execution loop
Batch execution completed 100 times.
```

Run status query

RESULT

Results | Messages

	Name	INDEXTYPE	INDEX_ID	DEPTH	IND_LEVEL	RECORDCOUNT	PageCount	Fragmentation
1	Indexing	HEAP	0	1	0	103	103	7

Create a clustered index to move from Heap to Custered

SQL

```sql
CREATE CLUSTERED INDEX CI_Indexing on Indexing(ID)
GO
```

RESULT

Messages

```
Command(s) completed successfully.
```

RESULT

Results | Messages

	Name	INDEXTYPE	INDEX_ID	DEPTH	IND_LEVEL	RECORDCOUNT	PageCount	Fragmentation
1	Indexing	CLUSTERED INDEX	1	2	0	103	103	1
2	Indexing	CLUSTERED INDEX	1	2	1	103	1	1

Query... | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | AdventureWorks2012 | 00:00:00 | 2 rows

Index type not clustered index
Index_id=1
Depth level now 2
103 records at depth level of 0
Another page with 103 records but page count of 1 used to manage these records
Add 700 more rows

SQL

```sql
insert into indexing values
 ('more junk', 'more junk companies', 100)
go 700
```

RESULT

Messages

```
Beginning execution loop
Batch execution completed 700 times.
```

RESULT

Results | Messages

	Name	INDEXTYPE	INDEX_ID	DEPTH	IND_LEVEL	RECORDCOUNT	PageCount	Fragmentation
1	Indexing	CLUSTERED INDEX	1	3	0	803	803	9
2	Indexing	CLUSTERED INDEX	1	3	1	803	2	2
3	Indexing	CLUSTERED INDEX	1	3	2	2	1	1

Query... | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | AdventureWorks2012 | 00:00:00 | 3 rows

803 records managed by 2 pages in intermediate row and 1 page in root node

Look at the statistics

```
-- Statistics speaks a lot !!!
DBCC SHOW_STATISTICS ('Indexing', CI_IndexingID)
GO
```

100 %

Results | Messages

	Name	Updated	Rows	Rows Sampl...	Ste...	Dens...	Average key len...	String Ind...	Filter Expressi...	Unfiltere
1	CI_IndexingID	Feb 26 2012 11:08AM	103	103	52	1	4	NO	NULL	103

	All density	Average Len...	Columns
1	0.009708738	4	ID

	RANGE_HI_K...	RANGE_RO...	EQ_RO...	DISTINCT_RANGE_RO...	AVG_RANGE_RO...
1	1	0	1	0	1
2	3	1	1	1	1
3	5	1	1	1	1
4	7	1	1	1	1
5	9	1	1	1	1
6	11	1	1	1	1
7	13	1	1	1	1
8	15	1	1	1	1
9	17	1	1	1	1

Distinct values are important so you can query
Create nonclustered on top of this

SQL

```
CREATE NONCLUSTERED INDEX NCI_PAY on Indexing(Pay)
Go
```

RESULT

Messages

```
Command(s) completed successfully.
```

Created on the pay column
Run status again

RESULT

Results | Messages

	Name	INDEXTYPE	INDEX_ID	DEPTH	IND_LEVEL	RECORDCOUNT	PageCount	Fragmentation
1	Indexing	CLUSTERED INDEX	1	3	0	803	803	11
2	Indexing	CLUSTERED INDEX	1	3	1	803	2	2
3	Indexing	CLUSTERED INDEX	1	3	2	2	1	1
4	Indexing	NONCLUSTERED ...	2	2	0	803	2	2
5	Indexing	NONCLUSTERED ...	2	2	1	2	1	1

Query executed suc... | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | tempdb | 00:00:00 | 5 rows

Non-clustered index show on 4th and 5th row
Slightly different from clustered index above
Only 2 pages – non-clustered index stored only the key and not the data

Makes pointer to the data
Show statistics

SQL

```
DBCC SHOW_STATISTICS('Indexing',NCI_PAY)
```

RESULT

Results | Messages

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows
1	NCI_PAY	Oct 9 2015 1:23PM	803	803	5	0	8	NO	NULL	803

	All density	Average Length	Columns
1	0.2	4	Pay
2	0.00124533	8	Pay, ID

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	100	0	700	0	1
2	1000	0	100	0	1
3	10000	0	1	0	1
4	13000	0	1	0	1
5	15000	0	1	0	1

Query executed successfully. | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | tempdb | 00:00:00 | 1 rows

ID is part of the pointer
Leaf node has a pointer of rows data of physical data

Additional Information

- Indexed views have the same storage structure as clustered tables

Facts:

- Fill factor value of 50 percent can cause database read performance to degrade by 2 times
- Non-clustered index per table-999
- Columns keys per index-16
- Statistics on non-indexed cvolumns-30,000
- XML indexes-249
- Maximum bytes in any index key-900bytes

Practical Indexing techniques

- Primary key
- Over Indexing
- Duplicate Index
- Clustered Index
- Unique Index

Primary key

- Primary Key automatically creates clustered index
  - Except-when non-clustered index is specified
  - Except when clustered index already exists

- Primary key automatically creates unique index on column
  - Non-Unique clustered index add 4 byte unique identifier column

SQL

```
use AdventureWorks2012
go
set nocount on
--create new table empty table
select * into sales.pksalesorderdetail
from Sales.SalesOrderDetail
where 1=2
go
```

RESULT

Messages

```
Command(s) completed successfully.
```

Insert data from salesorderdetail

SQL

```
insert into sales.pksalesorderdetail
(SalesOrderID, CarrierTrackingNumber, OrderQty, ProductID,
SpecialOfferID,UnitPrice,
UnitPriceDiscount,LineTotal, rowguid, ModifiedDate)
select salesorderid, CarrierTrackingNumber, OrderQty,
ProductID, SpecialOfferID, UnitPrice,
UnitPriceDiscount,LineTotal, rowguid, ModifiedDate
from sales.SalesOrderDetail
go
select * from sales.pksalesorderdetail
```

RESULT

100 %

Results | Messages

	SalesOrderID	SalesOrderDetailID	CarrierTrackingNumber	OrderQty	ProductID	SpecialOfferID	UnitPrice	UnitPriceDiscount	LineTotal	rowguid
1	43666	72	D46A-40CA-8D	1	753	1	2146.962	0.00	2146.962000	6422063E
2	43666	73	D46A-40CA-8D	1	732	1	356.898	0.00	356.898000	E992A08
3	43666	74	D46A-40CA-8D	1	756	1	874.794	0.00	874.794000	2A447585
4	43666	75	D46A-40CA-8D	2	768	1	419.4589	0.00	838.917800	F919E665
5	43666	76	D46A-40CA-8D	1	766	1	419.4589	0.00	419.458900	92A93F8
6	43667	77	4DFB-4B10-A6	3	710	1	5.70	0.00	17.100000	A3F2749
7	43667	78	4DFB-4B10-A6	1	773	1	2039.994	0.00	2039.994000	359B075
8	43667	79	4DFB-4B10-A6	1	778	1	2024.994	0.00	2024.994000	14A291B
9	43667	80	4DFB-4B10-A6	1	775	1	2024.994	0.00	2024.994000	151DDEE
10	43668	81	365D-4C9A-BE	3	756	1	874.794	0.00	2624.382000	10A9A12

Query executed successfully. | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | AdventureWorks2012 | 00:00:02 | 121317 rows

Now alter the database

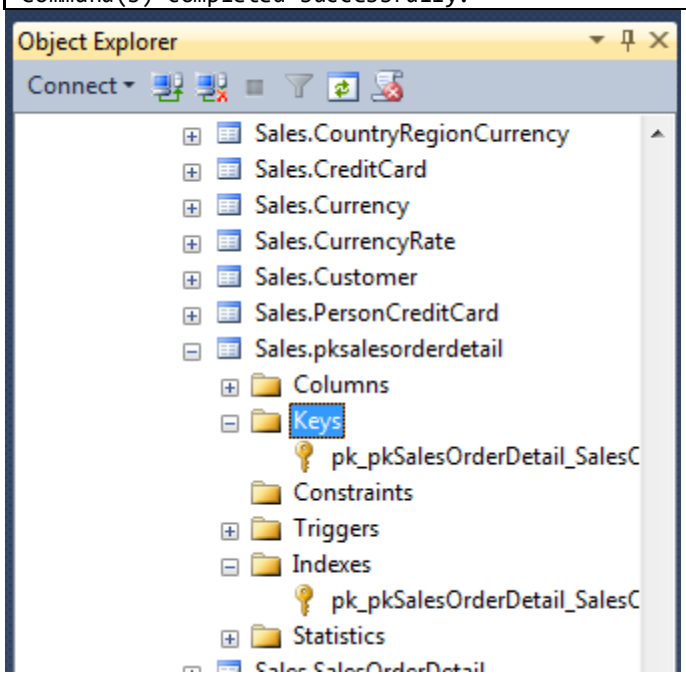
Create a primary key with clustered index not specified

SQL

```
alter table sales.pksalesorderdetail
add constraint pk_pkSalesOrderDetail_SalesOrderDetailID
Primary key(salesorderdetailid ASC )
go
```

RESULT

```
Command(s) completed successfully.
```



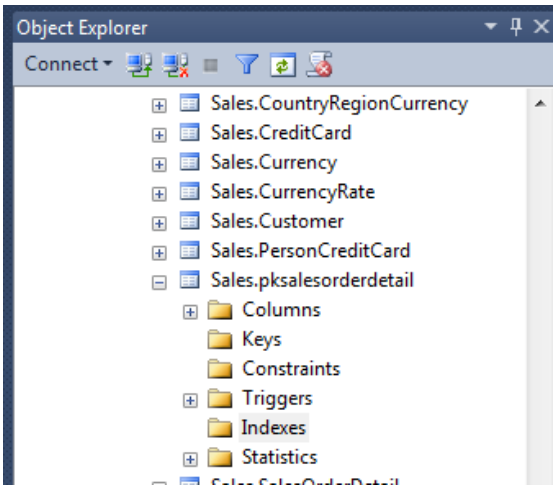
Makes a primary key and an index
Now drop the primarykey constraint

SQL

```
alter table sales.pksalesorderdetail
drop constraint pk_pksalesorderdetail_salesOrderDetailId
go
```

RESULT

```
Command(s) completed successfully.
```

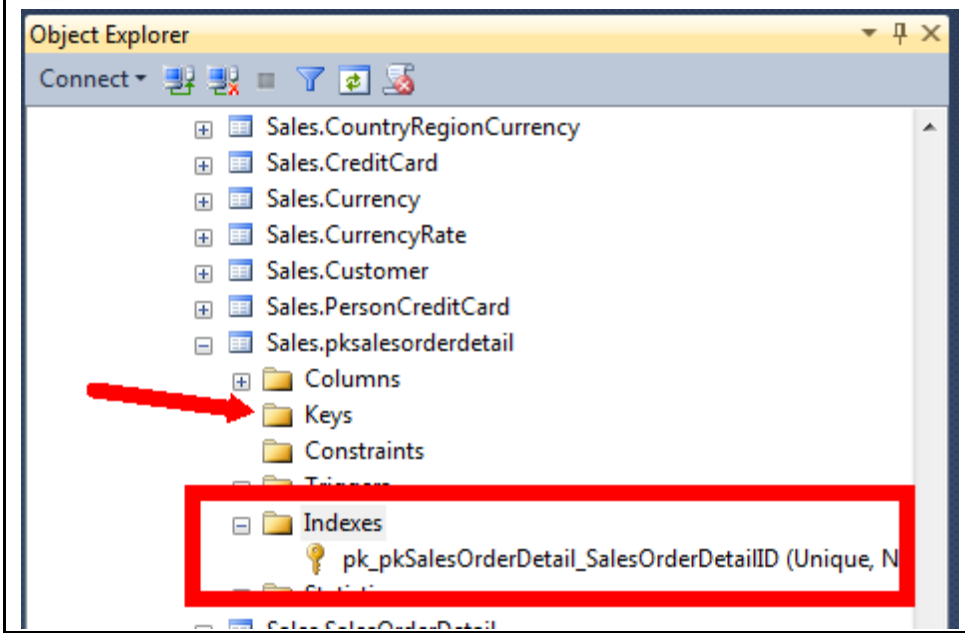


Now add a non-clustered primary key

SQL

```
alter table sales.pksalesorderdetail
add constraint pk_pkSalesOrderDetail_SalesOrderDetailID
Primary key nonclustered(salesorderdetailid ASC )
go
```

RESULT

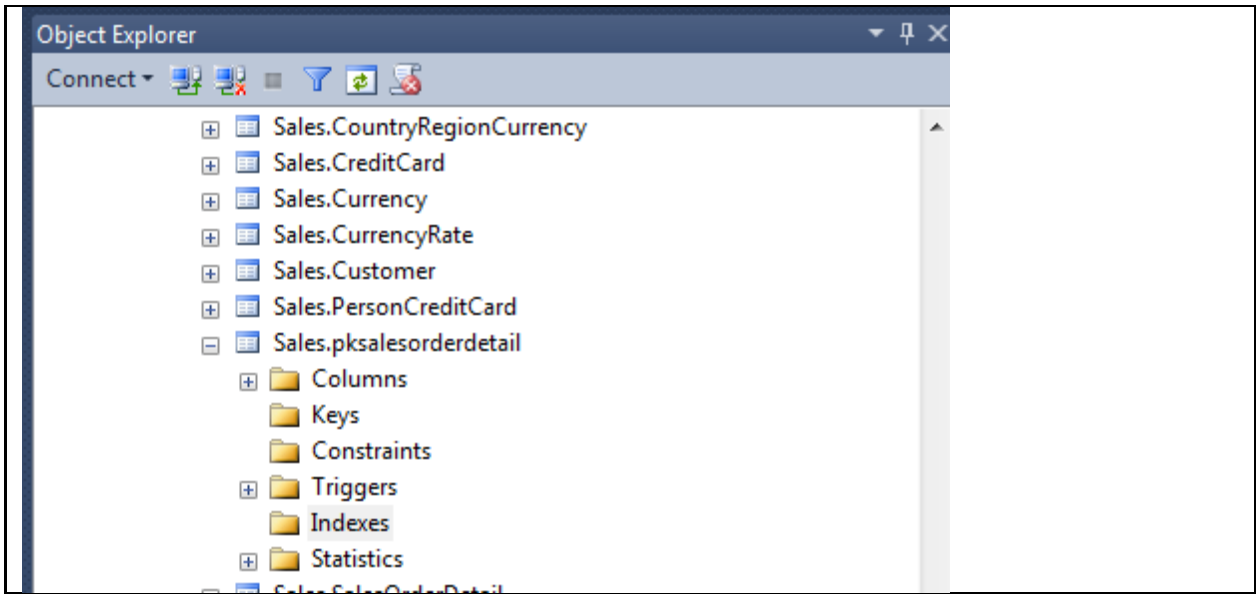


Drop the constraint

SQL

```
alter table sales.pksalesorderdetail
drop constraint pk_pksalesorderdetail_salesOrderDetailId
go
```

RESULT

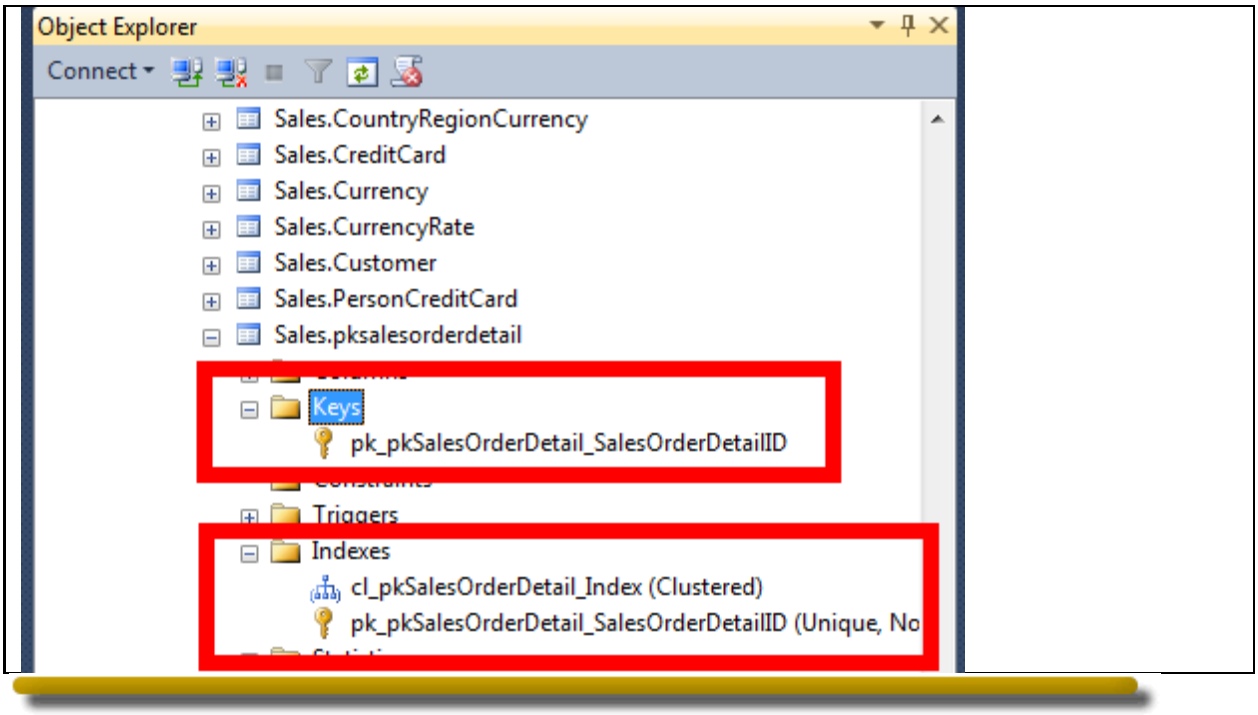


- Now create a clustered constraint
- Then create a primary key

SQL

```
create clustered index
cl_pkSalesOrderDetail_Index on sales.pksalesorderdetail
(
     salesorderid asc,
     carriertrackingNumber asc
)
Go

alter table sales.pksalesorderdetail
add constraint pk_pkSalesOrderDetail_SalesOrderDetailID
Primary key(salesorderdetailid ASC )
go
```

RESULT



Over indexing

- Consumes unnecessary disk space
- Queries may use less efficient index
- Less efficient execution plan
- Reduction in overall server performance
- Confusion among developers when troubleshooting when too many indexes
- Best practice: Drop unused indexes

Demo

Create an empty table

SQL

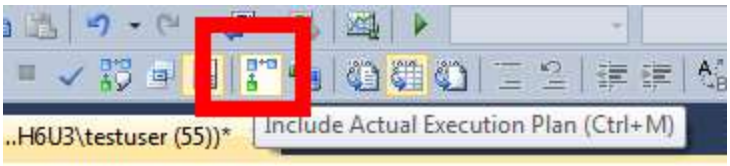
```
use adventureworks2012
GO

SET NOCOUNT ON

--create new empty table
SELECT *
INTO Sales.newsalesorderDetail
from Sales.SalesOrderDetail
WHERE 1=2
GO
```

Measure the Time

SQL

```
SET STATISTICS TIME ON
GO
```

Click ***Include Actual Execution Plan***



Insert some data

```
SQL

INSERT INTO Sales.newsalesorderDetail
(salesorderid, carriertrackingNumber, orderqty,
productID,specialOfferID, unitPrice,
UnitPriceDiscount, LineTotal, rowGuid, modifiedDate)
SELECT SalesOrderID, CarriertrackingNumber , orderQty,
ProductID,SpecialOfferID, UnitPrice,
UnitPriceDiscount, LineTOtal, rowGuid, ModifiedDate
FROM sales.salesorderdetail
GO
```

```
Messages | Execution plan
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 312 ms,  elapsed time = 672 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
```

Time gives you how long it took to execute
No index at this time
Then truncate the table

```
SQL

TRUNCATE TABLE Sales.newSalesOrderDetail
Go
```

Add clustered index and a number of nonclustered indexes

```
SQL

--Create Clustered Index
Alter Table Sales.newSalesOrderDetail
Add Constraint
PK_NewSalesOrderDetail_SalesOrderID_NewSalesOrderDetailID
Primary key Clustered
```

```
(SalesOrderID ASC,
SalesOrderDetailID ASC) ON [PRIMARY]
Go
--Create Non-Clustered Index
Create NONCLUSTERED INDEX
IX_NewSalesOrderDetail_CarrierTrackingNumber
ON Sales.NewSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
--Create Non-Clustered Index
Create NONCLUSTERED INDEX IX_NewSalesOrderDetail_OrderQty
ON Sales.NewSalesOrderDetail
(OrderQty ASC) on [PRIMARY]
Create NONCLUSTERED INDEX IX_NewSalesOrderDetail_ProductID
ON Sales.NewSalesOrderDetail
(ProductID ASC) on [PRIMARY]
Create NONCLUSTERED INDEX IX_NewSalesOrderDetail_SpecialOfferID
ON Sales.NewSalesOrderDetail
(SpecialOfferID ASC) on [PRIMARY]
Create NONCLUSTERED INDEX IX_NewSalesOrderDetail_UnitPrice
ON Sales.NewSalesOrderDetail
(UnitPrice ASC) on [PRIMARY]
Create NONCLUSTERED INDEX
IX_NewSalesOrderDetail_UnitPriceDiscount
ON Sales.NewSalesOrderDetail
(UnitPriceDiscount ASC) on [PRIMARY]
Create NONCLUSTERED INDEX IX_NewSalesOrderDetail_LineTotal
ON Sales.NewSalesOrderDetail
(LineTotal ASC) on [PRIMARY]
Create NONCLUSTERED INDEX IX_NewSalesOrderDetail_rowGuid
ON Sales.NewSalesOrderDetail
(rowGuid ASC) on [PRIMARY]
Create NONCLUSTERED INDEX IX_NewSalesOrderDetail_ModifiedDate
ON Sales.NewSalesOrderDetail
(ModifiedDate ASC) on [PRIMARY]
Create NONCLUSTERED INDEX IX_NewSalesOrderDetail_SpecialOfferID
ON Sales.NewSalesOrderDetail
(SpecialOfferID ASC) on [PRIMARY]
```

Now do another INSERT test

SQL

```
--Insert test again
INSERT INTO Sales.NewSalesOrderDetail
(SalesOrderID, CarrierTrackingNumber, OrderQty,
ProductID,SpecialOfferID, UnitPrice,
UnitPriceDiscount, LineTotal, rowGuid, ModifiedDate)
SELECT SalesOrderID, CarrierTrackingNumber, OrderQty,
ProductID,SpecialOfferID, UnitPrice,
```

```
UnitPriceDiscount, LineTotal, rowGuid, ModifiedDate
FROM Sales.SalesOrderDetail
Go
```

RESULT

Messages | Execution plan

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 14 ms.

 SQL Server Execution Times:
   CPU time = 561 ms,  elapsed time = 1613 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
```

Takes much longer to execute

SQL

```
drop table Sales.newsalesorderDetail
GO
```

Duplicate Index

- Reduces the performance of INSERT, UPDATE, DELETE Query
- No performance advantages to SELECTs
- Wasteful of space
- Best Practice: Drop Duplicate Indexes

Create a dummy table

SQL

```
use adventureworks2012
GO

SET NOCOUNT ON

--create new empty table
SELECT *
INTO Sales.DupsalesorderDetail
```

```
from Sales.SalesOrderDetail
WHERE 1=2
GO
```

- Measure the time

SQL

```
SET STATISTICS TIME ON
SET STATISTICS IO ON
GO
```

- Add 5 records

SQL

```
INSERT INTO Sales.DupSalesOrderDetail
(salesorderid, carriertrackingNumber, orderqty,
productID,specialOfferID, unitPrice,
UnitPriceDiscount, LineTotal, rowGuid, modifiedDate)
SELECT SalesOrderID, CarriertrackingNumber , orderQty,
ProductID,SpecialOfferID, UnitPrice,
UnitPriceDiscount, LineTotal, rowGuid, ModifiedDate
FROM sales.salesorderdetail
GO 5
```

RESULT

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 8 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
Beginning execution loop
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 250 ms.
Table 'DupsalesorderDetail'. Scan count 0, logical reads 122810, physical reads 0, read-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 780 ms,  elapsed time = 3037 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'DupsalesorderDetail'. Scan count 0, logical reads 122811, physical reads 0, read-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 655 ms,  elapsed time = 2694 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'DupsalesorderDetail'. Scan count 0, logical reads 122811, physical reads 0, read-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

```
 SQL Server Execution Times:
   CPU time = 920 ms,  elapsed time = 3621 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'DupsalesorderDetail'. Scan count 0, logical reads 122811, physical reads 0, read-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 874 ms,  elapsed time = 2844 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'DupsalesorderDetail'. Scan count 0, logical reads 122811, physical reads 0, read-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 920 ms,  elapsed time = 3909 ms.
Batch execution completed 5 times.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
```

- Truncate the table (Which removes all rows from the table without logging it Like a ***delete*** with no ***where*** but faster

SQL

```
TRUNCATE TABLE Sales.DupSalesOrderDetail
GO
```

- Add a bunch of duplicate indexes based on carriertracking number

SQL

```
Create Nonclustered Index
IX_NewSalesOrderDetail_CarrierTrackingNumber1
ON Sales.DupSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
Go
Create Nonclustered Index
IX_NewSalesOrderDetail_CarrierTrackingNumber2
ON Sales.DupSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
Go
Create Nonclustered Index
IX_NewSalesOrderDetail_CarrierTrackingNumber3
ON Sales.DupSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
Go
```

```
Create Nonclustered Index
IX_NewSalesOrderDetail_CarrierTrackingNumber4
ON Sales.DupSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
Go
Create Nonclustered Index
IX_NewSalesOrderDetail_CarrierTrackingNumber5
ON Sales.DupSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
Go
Create Nonclustered Index
IX_NewSalesOrderDetail_CarrierTrackingNumber6
ON Sales.DupSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
Go
Create Nonclustered Index
IX_NewSalesOrderDetail_CarrierTrackingNumber7
ON Sales.DupSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
Go
Create Nonclustered Index
IX_NewSalesOrderDetail_CarrierTrackingNumber8
ON Sales.DupSalesOrderDetail
(CarrierTrackingNumber ASC) on [PRIMARY]
Go
```

- Do another insert and measure the time difference

SQL

```
INSERT INTO Sales.DupsalesorderDetail
(SalesOrderID,CarrierTrackingNumber,OrderQty,
ProductID,SpecialOfferID,UnitPrice,
UnitPriceDiscount,LineTotal,rowguid,ModifiedDate)
SELECT SalesOrderID,CarrierTrackingNumber,OrderQty,
ProductID,SpecialOfferID,UnitPrice,
UnitPriceDiscount,LineTotal,rowguid,ModifiedDate
FROM Sales.SalesOrderDetail
Go 5
```

RESULT

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
Beginning execution loop
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 225 ms.
Table 'DupsalesorderDetail'. Scan count 0, logical reads 2376882, physical reads 0, read-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'Worktable'. Scan count 8, logical reads 274713, physical reads 0, read-ahead reads 0,
lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

```
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 11903 ms,  elapsed time = 17013 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'DupsalesorderDetail'. Scan count 0, logical reads 3080427, physical reads 0, read-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'Worktable'. Scan count 8, logical reads 274713, physical reads 0, read-ahead reads 0,
lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 14508 ms,  elapsed time = 19899 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'DupsalesorderDetail'. Scan count 0, logical reads 3088611, physical reads 0, read-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'Worktable'. Scan count 8, logical reads 274713, physical reads 0, read-ahead reads 0,
lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 12870 ms,  elapsed time = 22613 ms.
```

The logical reads are multiplied as is the time to execute

The execution plan is crazy complicated now
Took 46 seconds
At least 11-12 times slower to table with NO indexes
Clean up your table

SQL

```sql
DROP TABLE Sales.DupsalesorderDetail
GO
```

Clustered index

- Determines the physical orders of the data in the table
- Improves the performance of
  - Columns that contain a large number of distinct values
  - Columns that are accessed frequently
  - Columns that are often searched for ranges of values
  - Queries returning huge result sets
- Best Practices
  - Create Clustered Indexes when necessary
  - Avoid wide keys
  - Avoid Creating columns that are frequently changing
- Note: In SQL Server – Primary Keys automatically created a clustered index

DEMO

SQL

```sql
use adventureworks2012
GO

SET NOCOUNT ON

--create new empty table
SELECT *
INTO Sales.MySalesOrderDetail
from Sales.SalesOrderDetail
WHERE 1=2
GO

SET STATISTICS TIME ON
SET STATISTICS IO ON
GO
```

- Now insert 100,000 rows
  - By adding 10,000 rows 10 times

SQL

```sql
INSERT INTO Sales.MySalesOrderDetail
(salesorderid, carriertrackingNumber, orderqty,
productID,specialOfferID, unitPrice,
UnitPriceDiscount, LineTotal, rowGuid, modifiedDate)
SELECT SalesOrderID, CarriertrackingNumber , orderQty,
ProductID,SpecialOfferID, UnitPrice,
UnitPriceDiscount, LineTotal, rowGuid, ModifiedDate
FROM sales.salesorderdetail
GO 5
```

RESULT

```
Beginning execution loop
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 33 ms.
Table 'MySalesOrderDetail'. Scan count 0, logical reads 122810, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 1747 ms,  elapsed time = 2751 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 1 ms.
Table 'MySalesOrderDetail'. Scan count 0, logical reads 122811, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 2293 ms,  elapsed time = 3342 ms.
```

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'MySalesOrderDetail'. Scan count 0, logical reads 122811, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 2278 ms,  elapsed time = 3137 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'MySalesOrderDetail'. Scan count 0, logical reads 122811, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 842 ms,  elapsed time = 1541 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'MySalesOrderDetail'. Scan count 0, logical reads 122811, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'SalesOrderDetail'. Scan count 1, logical reads 1246, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 1186 ms,  elapsed time = 1652 ms.
Batch execution completed 5 times.
```

This table has no Indexes

- Run the following query

SQL

```
SELECT *
FROM Sales.MySalesOrderDetail sod
WHERE sod.SalesOrderDetailID>40000
and sod.SalesOrderDetailID<160000
```

RESULT

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 48 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'MySalesOrderDetail'. Scan count 1, logical reads 7470, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 265 ms,  elapsed time = 2432 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
```

Brings 7470 reads

- Create a primary key clustered index

SQL

```
ALTER TABLE Sales.MySalesOrderDetail
Add Constraint PK_MySlaesOrderDetail_SalesOderDetailID
Primary Key Clustered
(SalesOrderDetailID ASC) ON [PRIMARY]
```

- Run the same query

SQL

```
SELECT *
FROM Sales.MySalesOrderDetail sod
WHERE sod.SalesOrderDetailID>40000
and sod.SalesOrderDetailID<160000
```

RESULT

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 77 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'MySalesOrderDetail'. Scan count 1, logical reads 1484, physical reads 0, read-ahead
reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 219 ms,  elapsed time = 2545 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
```

Logical reads are now 1484

Io reads much less
1/10
And 10 times faster
Clean up your table

SQL

```
DROP TABLE Sales.MySalesOrderDetail
Go
```

Unique Index

- Enforces uniqueness in column
- Unique constraint is same as unique index
- Often improves performance of grouped by and aggregated queries
- Best Practice: Create unique index when business needs unique values in the column

DEMO
Add the following query
Make sure to turn on ***Include actual execution plan***

SQL

```
use adventureworks2012
GO

SET NOCOUNT ON

--create new empty table
SELECT *
INTO Sales.UniSalesOrderDetail
from Sales.SalesOrderDetail
WHERE 1=2
GO

SET STATISTICS TIME ON
SET STATISTICS IO ON
GO

INSERT INTO Sales.UniSalesOrderDetail
(salesorderid, carriertrackingNumber, orderqty,
productID,specialOfferID, unitPrice,
UnitPriceDiscount, LineTotal, rowGuid, modifiedDate)
SELECT SalesOrderID, CarriertrackingNumber , orderQty,
ProductID,SpecialOfferID, UnitPrice,
UnitPriceDiscount, LineTotal, rowGuid, ModifiedDate
FROM sales.salesorderdetail
GO 5
```

- Add the following Select query

SQL

```
SELECT SalesOrderDetailID
FROM Sales.UniSalesOrderDetail
Go
SELECT DISTINCT SalesOrderDetailID
```

```
FROM Sales.UniSalesOrderDetail
Go
```

RESULT

Results | Messages

	SalesOrderDetailID
1	72
2	73
3	74
4	75

	SalesOrderDetailID
1	51201
2	113548
3	175895
4	238242
5	300589
6	351790
7	414137
8	476484

Query executed succe... | WIN-A7M6IOSH6U3\SQLSERVER20... | WIN-A7M6IOSH6U3\testus... | AdventureWorks2012 | 00:00:11 | 606585 rows

Results | Messages | Execution plan

Query 1: Query cost (relative to the batch): 30%
SELECT SalesOrderDetailID FROM Sales.UniSalesOrderDetail

SELECT
Cost: 0 %

Table Scan
[UniSalesOrderDetail]
Cost: 100 %

Query 2: Query cost (relative to the batch): 70%
SELECT DISTINCT SalesOrderDetailID FROM Sales.UniSalesOrderDetail

SELECT
Cost: 0 %

Hash Match
(Aggregate)
Cost: 57 %

Table Scan
[UniSalesOrderDetail]
Cost: 43 %

- Click execution plan

Second query cost more to run
30% vs 70%

- Add unique nonclustered index and run again

SQL

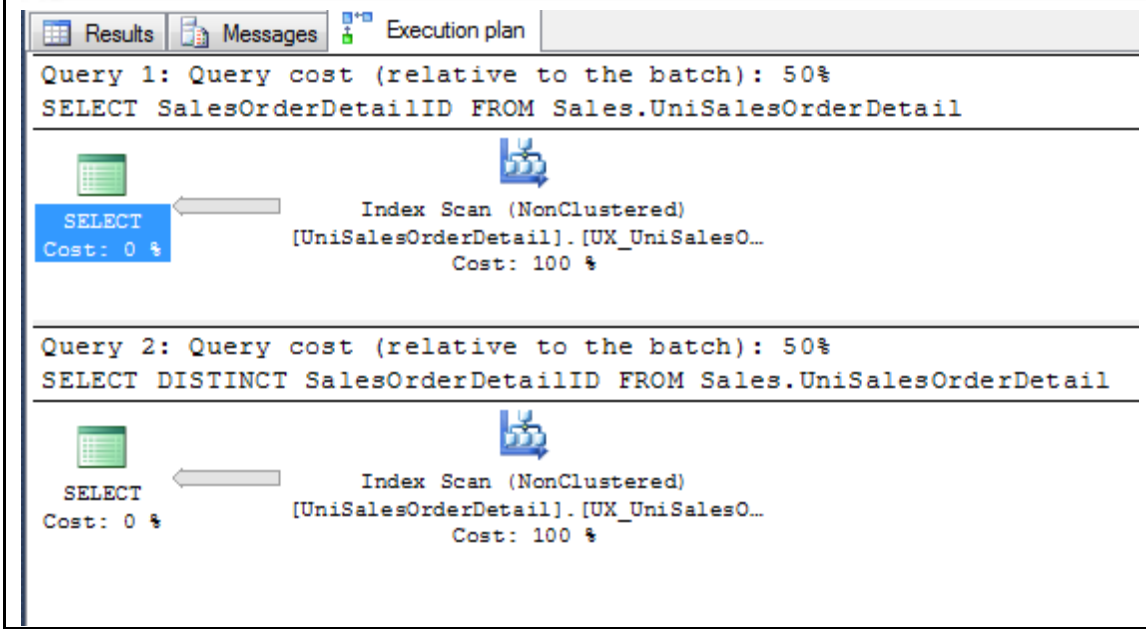
```
ALTER TABLE Sales.unisalesorderdetail
Add Constraint UX_UniSalesOrderDetail_SalesOrderDetailID
UNIQUE NONCLUSTERED
(SalesOrderDetailID) ON [PRIMARY]
GO
```

- Run the select query again

SQL

```
SELECT SalesOrderDetailID
FROM Sales.UniSalesOrderDetail
Go
SELECT DISTINCT SalesOrderDetailID
FROM Sales.UniSalesOrderDetail
Go
```

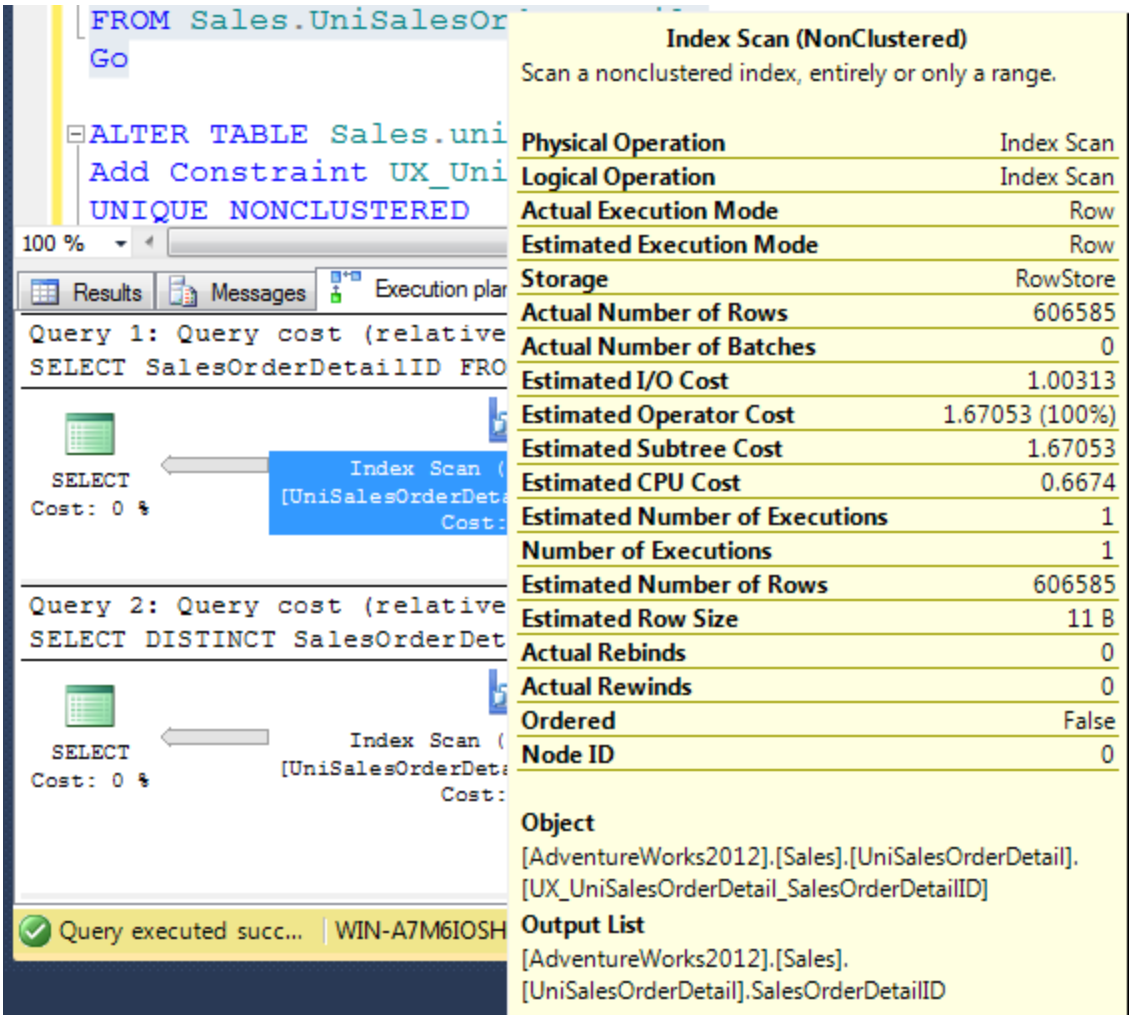
RESULT



- Click on Execution Plan

Both are showing same value of index nonclustered scan

- Click on one of the nodes and you get more detailed information



Query cost is now 50% on each

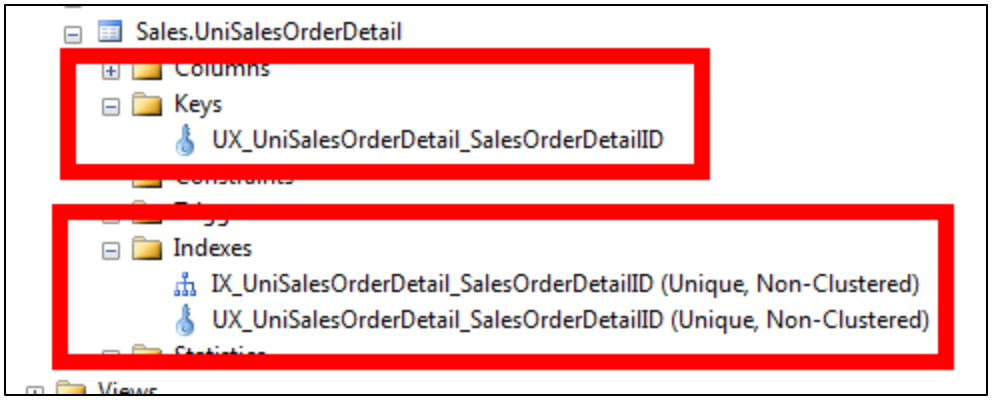
Create unique nonclustered index

Instead of constraint

SQL

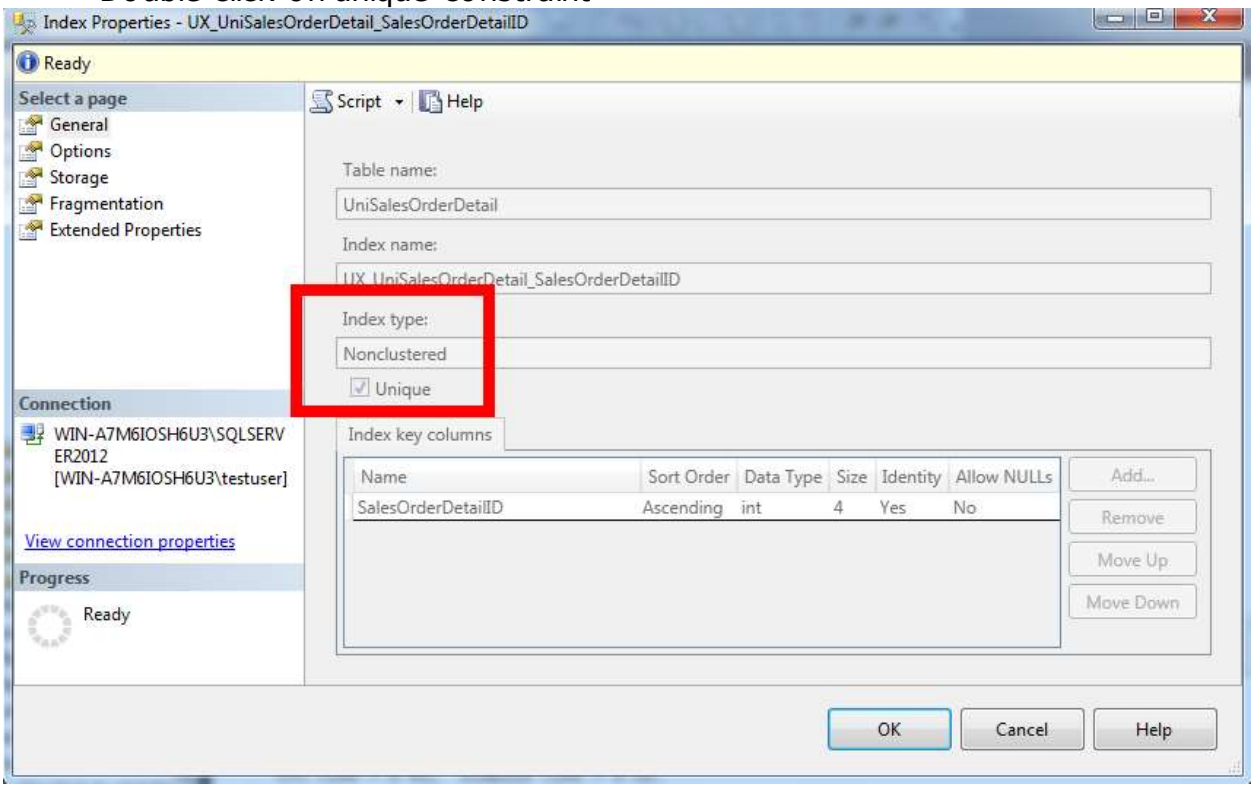
```
Create Unique Nonclustered index
IX_UniSalesOrderDetail_SalesOrderDetailID
ON Sales.UniSalesOrderDetail
(SalesOrderDetailID)
ON [Primary]
Go
```

Refresh your tables and look at them in object explorer

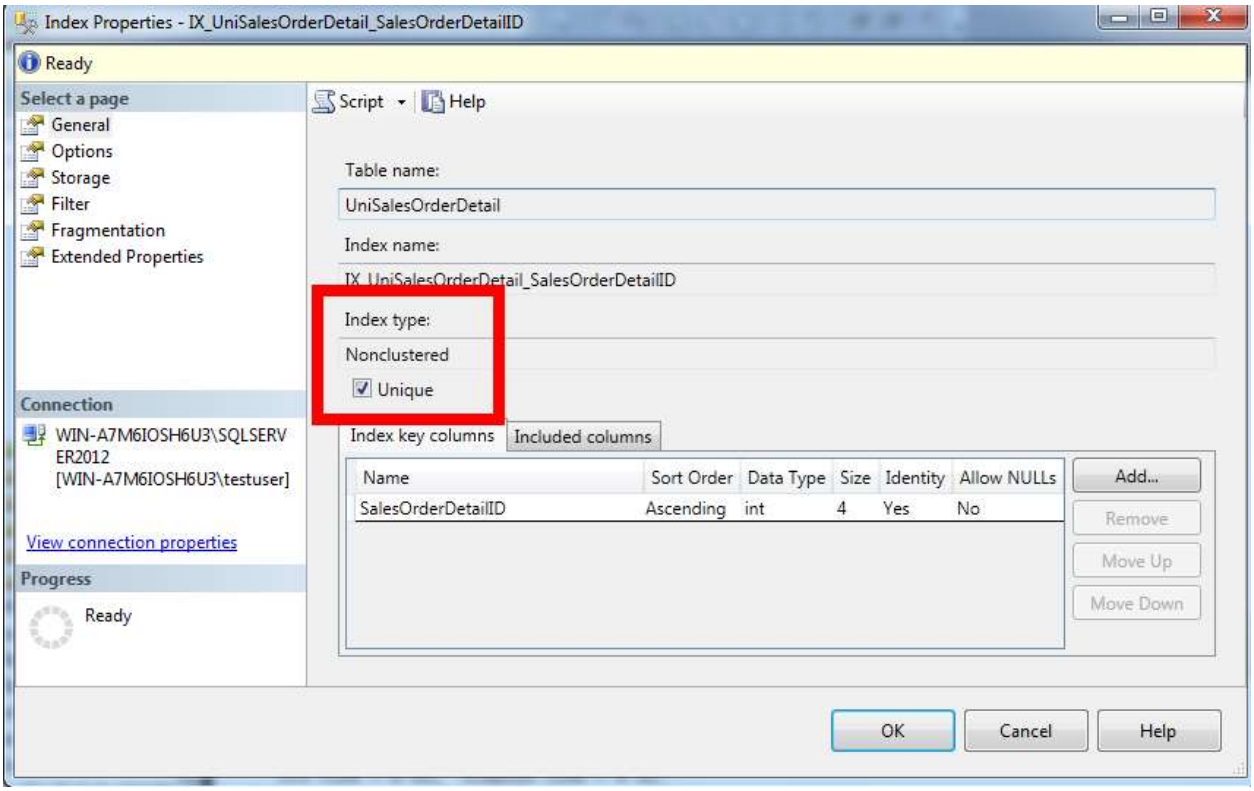


1st is an index
2nd is a key

- Double click on unique constraint



- Double click on other one and get the same



However, you can uncheck the uniqueness

Cannot change index defined by constraint but can for a nonclustered index

Clean up

```
Drop Table Sales.UniSalesOrderDetail
Go
```