INDEXES

An index is a performance-tuning method of allowing faster retrieval of records. An index creates an entry for each value that appears in the indexed columns. An index creates a relational index on a table or view.

A **rowstore index** is used to improve query performance, especially when the queries select from specific columns or require values to be sorted in a particular order. This index can be created before there is data in the table.

Unique - Creates a unique index on a table or view. A unique index is one in which no two rows are permitted to have the same index key value. A clustered index on a view must be unique. The Database Engine does not allow creating a unique index on columns that already include duplicate values, whether or not IGNORE_DUP_KEY is set to ON. The Database Engine checks for duplicate values each time data is added by a insert operations. Insert operations that would generate duplicate key values are rolled back, and the Database Engine displays an error message. Columns that are used in a unique index should be set to NOT NULL. This index indicates that the combination of values in the indexed columns must be unique.

Clustered - Creates an index in which the logical order of the key values determines the physical order of the corresponding rows in a table. The bottom, or leaf, level of the clustered index contains the actual data rows of the table. A table or view is allowed one clustered index at a time. A view with a unique clustered index is called an indexed view. Creating a unique clustered index on a view physically materializes the view. A unique clustered index must be created on a view before any other indexes can be defined on the same view. Create the clustered index before creating any nonclustered indexes. Existing nonclustered indexes on tables are rebuilt when a clustered index is created. If CLUSTERED is not specified, a nonclustered index is created.

Nonclustered - Creates an index that specifies the logical ordering of a table. With a nonclustered index, the physical order of the data rows is independent of their indexed order. Each table can have up to 999 nonclustered indexes, regardless of how the indexes are created: either implicitly with PRIMARY KEY and UNIQUE constraints, or explicitly with CREATE INDEX. For indexed views, nonclustered indexes can be created only on a view that has a unique clustered index already defined. If not otherwise specified, the default index type is nonclustered.

Partitioned Indexes - are created and maintained like the partitioned tables, but like ordinary indexes, they are handled as separate database objects. If you are creating an index on a partitioned table, and do not specify a filegroup on which to place the index, the index is partitioned in the same manner as the underlying table. When the index uses the same partition scheme and partitioning column as the table, the index is *aligned* with the table.

Filtered Indexes - is an optimized nonclustered index, suited for queries that select a small percentage of rows from a table. It uses a filter predicate to index a portion of the data in the table. A well-designed filtered index can improve query performance, reduce storage costs, and reduce maintenance costs.

Index Key Size - The maximum size for an index key is 900 bytes for a clustered index and 1,700 bytes for a nonclustered index. The index key of a clustered index cannot contain varchar columns that have existing data in the ROW_OVERFLOW_DATA allocation unit. Nonclustered indexes can include non-key columns in the leaf level of the index. These columns are not considered by the Database Engine when calculating the index key size.

Rename an Index: sp_rename 'table_name.old_index_name', 'new_index_name', 'INDEX'

```
-- rename an index
sp_rename 'contacts.contacts_idx', 'contacts_index_cname', 'INDEX';
-- renaming the index on the contacts table called contacts_idx to contacts_index_cname.
```

Drop an Index: DROP INDEX table_name.index_name;

Check the indexes: "Object Explorer-> Databases-> Database_Name-> Tables-> Table_Name -> Indexes" OR **EXECUTE sp_helpindex table_name**

```
use AdventureWorks2012; go
-- check the indexes
EXECUTE sp_helpindex [Sales.Currency]
```

Results | Messages

	index_name	index_description	index_keys
1	AK_Currency_Name	nonclustered, unique located on PRIMARY	Name
2	PK_Currency_CurrencyCode	clustered, unique, primary key located on PRIMARY	CurrencyCode

Computed Columns

Indexes can be created on computed columns. These ones can have the property PERSISTED (the Database Engine stores the computed values in the table, and updates them when any other columns on which the computed column depends are updated). The Database Engine uses these persisted values when it creates an index on the column, and when the index is referenced in a query. To index a computed column, the computed column must deterministic and precise.

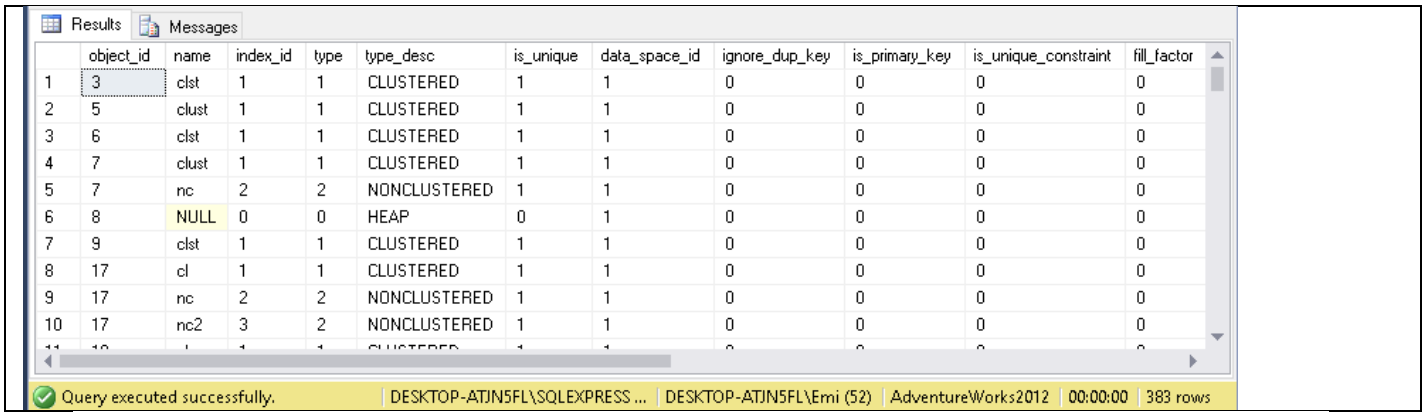
```
DROP TABLE t1
CREATE TABLE t1 (a int, b int, c AS a/b);
INSERT INTO t1 VALUES (1, 0);
select * from t1
-- after creating the table, you create an index on computed column c, the same
INSERTstatement will now fail.
drop table t1
CREATE TABLE t1 (a int, b int, c AS a/b);
CREATE UNIQUE CLUSTERED INDEX Idx1 ON t1(c);
INSERT INTO t1 VALUES (1, 0);
```

```
Msg 8134, Level 16, State 1, Line 13
Divide by zero error encountered.
The statement has been terminated.
```

Included Columns in Indexes

Non-key columns (or included columns), can be added to the leaf level of a nonclustered index to improve query performance by covering the query. This allows the query optimizer to locate all the required information from an index scan; the table or clustered index data is not accessed.

```
-- A. Create a simple nonclustered rowstore index
-- create a nonclustered index on the VendorID column of the Purchasing.ProductVendor table.
drop index IX_VendorID ON Production.WorkOrder
drop index IX_VendorID_1 ON Production.WorkOrder
drop index IX_VendorID_2 ON Purchasing.ProductVendor
CREATE INDEX IX_VendorID ON Production.WorkOrder (WorkOrderID);
CREATE INDEX IX_VendorID_1 ON Production.WorkOrder (WorkOrderID DESC, OrderQty ASC, DueDate DESC);
CREATE INDEX IX_VendorID_2 ON Purchasing.ProductVendor (ProductID);
select * from sys.indexes
```

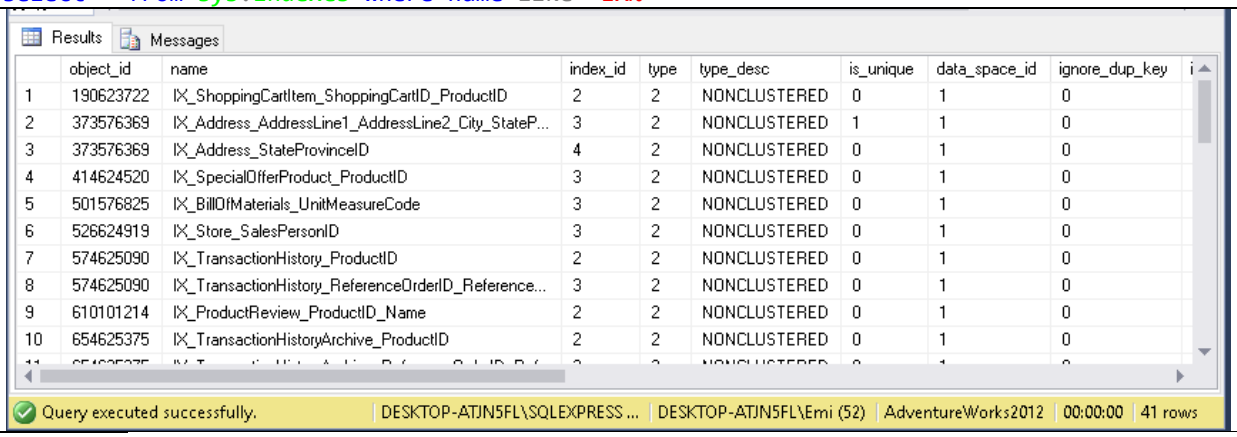


Results | Messages

	object_id	name	index_id	type	type_desc	is_unique	data_space_id	ignore_dup_key	is_primary_key	is_unique_constraint	fill_factor
1	3	clst	1	1	CLUSTERED	1	1	0	0	0	0
2	5	clust	1	1	CLUSTERED	1	1	0	0	0	0
3	6	clst	1	1	CLUSTERED	1	1	0	0	0	0
4	7	clust	1	1	CLUSTERED	1	1	0	0	0	0
5	7	nc	2	2	NONCLUSTERED	1	1	0	0	0	0
6	8	NULL	0	0	HEAP	0	1	0	0	0	0
7	9	clst	1	1	CLUSTERED	1	1	0	0	0	0
8	17	cl	1	1	CLUSTERED	1	1	0	0	0	0
9	17	nc	2	2	NONCLUSTERED	1	1	0	0	0	0
10	17	nc2	3	2	NONCLUSTERED	1	1	0	0	0	0

Query executed successfully. | DESKTOP-ATJN5FL\SQLEXPRESS ... | DESKTOP-ATJN5FL\Emi (52) | AdventureWorks2012 | 00:00:00 | 383 rows

```
-- B. Create a simple nonclustered rowstore composite index
-- creates a nonclustered composite index on the SalesQuota and SalesYTDcolumns of the Sales.SalesPerson
table.
CREATE NONCLUSTERED INDEX IX_SalesPerson_SalesQuota_SalesYTD ON Sales.SalesPerson (SalesQuota, SalesYTD);
select * from sys.indexes where name like 'IX%'
```



Results | Messages

	object_id	name	index_id	type	type_desc	is_unique	data_space_id	ignore_dup_key	i
1	190623722	IX_ShoppingCartItem_ShoppingCartID_ProductID	2	2	NONCLUSTERED	0	1	0	
2	373576369	IX_Address_AddressLine1_AddressLine2_City_StateP...	3	2	NONCLUSTERED	1	1	0	
3	373576369	IX_Address_StateProvinceID	4	2	NONCLUSTERED	0	1	0	
4	414624520	IX_SpecialOfferProduct_ProductID	3	2	NONCLUSTERED	0	1	0	
5	501576825	IX_BillOfMaterials_UnitMeasureCode	3	2	NONCLUSTERED	0	1	0	
6	526624919	IX_Store_SalesPersonID	3	2	NONCLUSTERED	0	1	0	
7	574625090	IX_TransactionHistory_ProductID	2	2	NONCLUSTERED	0	1	0	
8	574625090	IX_TransactionHistory_ReferenceOrderID_Reference...	3	2	NONCLUSTERED	0	1	0	
9	610101214	IX_ProductReview_ProductID_Name	2	2	NONCLUSTERED	0	1	0	
10	654625375	IX_TransactionHistoryArchive_ProductID	2	2	NONCLUSTERED	0	1	0	

Query executed successfully. | DESKTOP-ATJN5FL\SQLEXPRESS ... | DESKTOP-ATJN5FL\Emi (52) | AdventureWorks2012 | 00:00:00 | 41 rows

```
-- C. Create an index on a table in another database
-- creates a non-clustered index on the VendorID column of the ProductVendortable in the
Purchasing database.
CREATE CLUSTERED INDEX IX_ProductVendor_VendorID_3 ON Purchasing.ProductVendor (ProductID);
select * from sys.indexes where name like 'IX%'
```

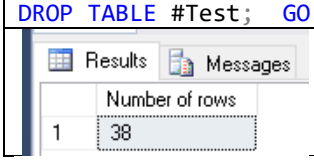
```
-- Add a column to an index
-- creates index IX_FF with two columns from the dbo.FactFinance table. The next
statement rebuilds the index with one more column and keeps the existing name.
CREATE INDEX IX_FF ON dbo.FactFinance ( FinanceKey ASC, DateKey ASC );
--Rebuild and add the OrganizationKey
CREATE INDEX IX_FF ON dbo.FactFinance ( FinanceKey, DateKey, OrganizationKey DESC)
WITH ( DROP_EXISTING = ON );
```

```
-- E. Create a unique nonclustered index
-- creates a unique nonclustered index on the Name column of the Production.UnitMeasure
table in the AdventureWorks2012 database. The index will enforce uniqueness on the data
inserted into the Name column.
CREATE UNIQUE INDEX AK_UnitMeasure_Name ON Production.UnitMeasure(Name);
-- tests the uniqueness constraint by attempting to insert a row with the same value as
that in an existing row. --Verify the existing value.
```

```
SELECT Name FROM Production.UnitMeasure WHERE Name = N'Ounces';  GO
INSERT INTO Production.UnitMeasure (UnitMeasureCode, Name, ModifiedDate)
    VALUES ('OC', 'Ounces', GetDate());
```

```
Msg 1913, Level 16, State 1, Line 54
The operation failed because an index or statistics with name 'AK_UnitMeasure_Name'
already exists on table 'Production.UnitMeasure'.
Msg 2601, Level 14, State 1, Line 60
Cannot insert duplicate key row in object 'Production.UnitMeasure' with unique index
'AK_UnitMeasure_Name'. The duplicate key value is (Ounces).
The statement has been terminated.
```

```
-- Use the IGNORE_DUP_KEY option
-- demonstrates the effect of the IGNORE_DUP_KEY option by inserting multiple rows into a
temporary table first with the option set to ON and again with the option set to OFF. A
single row is inserted into the #Test table that will intentionally cause a duplicate
value when the second multiple-row INSERT statement is executed. A count of rows in the
table returns the number of rows inserted.
CREATE TABLE #Test (C1 nvarchar(10), C2 nvarchar(50), C3 datetime); GO
CREATE UNIQUE INDEX AK_Index ON #Test (C2) WITH (IGNORE_DUP_KEY = ON); GO
INSERT INTO #Test VALUES (N'OC', N'Ounces', GETDATE());
INSERT INTO #Test SELECT * FROM Production.UnitMeasure;  GO
SELECT COUNT(*)AS [Number of rows] FROM #Test;  GO
DROP TABLE #Test;  GO
```



Notice that the rows inserted from the Production.UnitMeasure table that did not violate the uniqueness constraint were successfully inserted. A warning was issued and the duplicate row ignored, but the entire transaction was not rolled back.

```
-- The same statements are executed again, but with IGNORE_DUP_KEY set to OFF.
CREATE TABLE #Test (C1 nvarchar(10), C2 nvarchar(50), C3 datetime);  GO
CREATE UNIQUE INDEX AK_Index ON #Test (C2) WITH (IGNORE_DUP_KEY = OFF); GO
INSERT INTO #Test VALUES (N'OC', N'Ounces', GETDATE());
INSERT INTO #Test SELECT * FROM Production.UnitMeasure;  GO
SELECT COUNT(*)AS [Number of rows] FROM #Test;  GO
DROP TABLE #Test;  GO
```

```
(1 row(s) affected)
Msg 2601, Level 14, State 1, Line 89
Cannot insert duplicate key row in object 'dbo.#Test' with unique index 'AK_Index'. The
duplicate key value is (Ounces).
The statement has been terminated.
(1 row(s) affected)
```

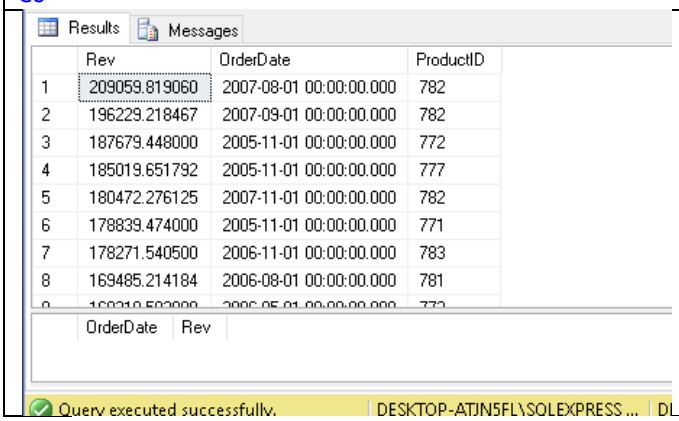
Notice that none of the rows from the Production.UnitMeasure table were inserted into the table even though only one row in the table violated the UNIQUE index constraint.

```
-- G. Using DROP_EXISTING to drop and re-create an index
-- drops and re-creates an existing index on the ProductID column of the Production.WorkOrder table in the
AdventureWorks2012 database by using the DROP_EXISTINGoption. The options FILLFACTOR and PAD_INDEX are
also set.
CREATE NONCLUSTERED INDEX IX_WorkOrder_ProductID ON Production.WorkOrder(ProductID)
    WITH (FILLFACTOR = 80,  PAD_INDEX = ON,  DROP_EXISTING = ON);
GO
```

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

```
-- H. Create an index on a view
```

```sql
-- creates a view and an index on that view. Two queries are included that use the indexed view.
--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
```




```sql
-- I. Create an index with included (non-key) columns
-- creates a nonclustered index with one key column (PostalCode) and four non-key columns
(AddressLine1, AddressLine2, City, StateProvinceID). A query that is covered by the index
follows. To display the index that is selected by the query optimizer, on the Query menu in SQL
Server Management Studio, select Display Actual Execution Plan before executing the query.
CREATE NONCLUSTERED INDEX IX_Address_PostalCode ON Person.Address (PostalCode)
    INCLUDE (AddressLine1, AddressLine2, City, StateProvinceID);
GO
```

```
SELECT AddressLine1, AddressLine2, City, StateProvinceID, PostalCode
FROM Person.Address
WHERE PostalCode BETWEEN N'98000' and N'99999';
GO
```

Results | Messages

	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode
1	225 South 314th Street	NULL	Federal Way	79	98003
2	8684 Military East	NULL	Bellevue	79	98004
3	7270 Pepper Way	NULL	Bellevue	79	98004
4	6058 Hill Street	# 4	Bellevue	79	98004
5	1648 Eastgate Lane	NULL	Bellevue	79	98004
6	3454 Bel Air Drive	NULL	Bellevue	79	98004
7	3067 Maya	NULL	Bellevue	79	98004
8	3197 Thornhill Place	NULL	Bellevue	79	98004
9	3919 Pinto Road	NULL	Bellevue	79	98004
10	7396 Stratton Circle	NULL	Bellevue	79	98004
11	9745 Bonita Ct.	NULL	Bellevue	79	98004

Query executed successfully. | DESKTOP-ATJN5FL\SQLEXPRESS ... | DESKTOP-ATJN5

```
-- J. Create a partitioned index
-- creates a nonclustered partitioned index on TransactionsPS1, an existing partition scheme in the
AdventureWorks2012 database. This example assumes the partitioned index sample has been installed.
CREATE NONCLUSTERED INDEX IX_TransactionHistory_ReferenceOrderID
    ON Production.TransactionHistory (ReferenceOrderID)
    ON TransactionsPS1 (TransactionDate);
GO
```

```
-- K. Creating a filtered index
-- creates a filtered index on the Production.BillOfMaterials table in the AdventureWorks2012
database. The filter predicate can include columns that are not key columns in the filtered index.
The predicate in this example selects only the rows where EndDate is non-NULL.
CREATE NONCLUSTERED INDEX "FIBillOfMaterialsWithEndDate" ON Production.BillOfMaterials (ComponentID,
StartDate) WHERE EndDate IS NOT NULL;
```

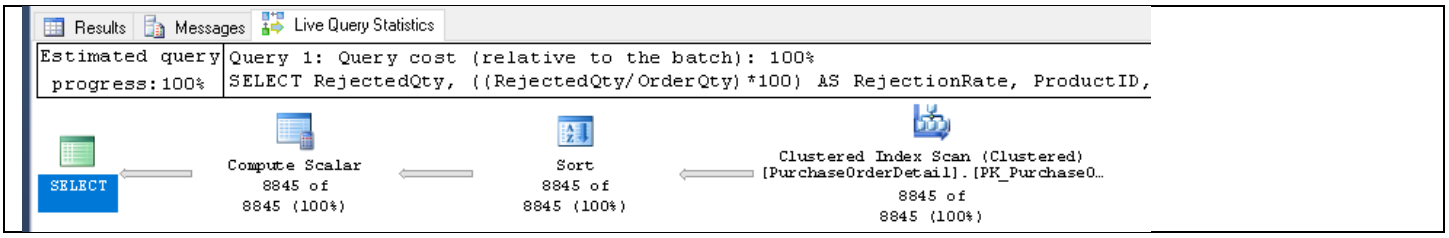
```
Command(s) completed successfully.
```

```
-- L. Create a compressed index
-- creates an index on a nonpartitioned table by using row compression.
CREATE NONCLUSTERED INDEX IX_INDEX_1 ON T1 (C2) WITH ( DATA_COMPRESSION = ROW ) ;  GO
```

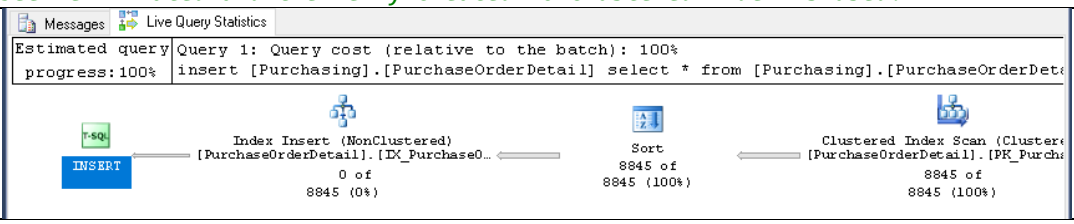
```
-- creates an index on a partitioned table by using row compression on all partitions of the index.
CREATE CLUSTERED INDEX IX_PartTab2Col1 ON PartitionTable1 (Col1) WITH ( DATA_COMPRESSION = ROW ) ;
GO
-- creates an index on a partitioned table by using page compression on partition 1 of the index and
row compression on partitions 2 through 4 of the index.
CREATE CLUSTERED INDEX IX_PartTab2Col1 ON PartitionTable1 (Col1)
WITH (DATA_COMPRESSION = PAGE ON PARTITIONS(1),
    DATA_COMPRESSION = ROW ON PARTITIONS (2 TO 4 ) ) ;  GO
```

Index Sort Order Design

```
-- execution plan - The following execution plan for this query shows that the query optimizer used a
SORT operator to return the result set in the order specified by the ORDER BY clause.
SELECT RejectedQty, ((RejectedQty/OrderQty)*100) AS RejectionRate, ProductID, DueDate
FROM Purchasing.PurchaseOrderDetail
ORDER BY RejectedQty DESC, ProductID ASC;
```

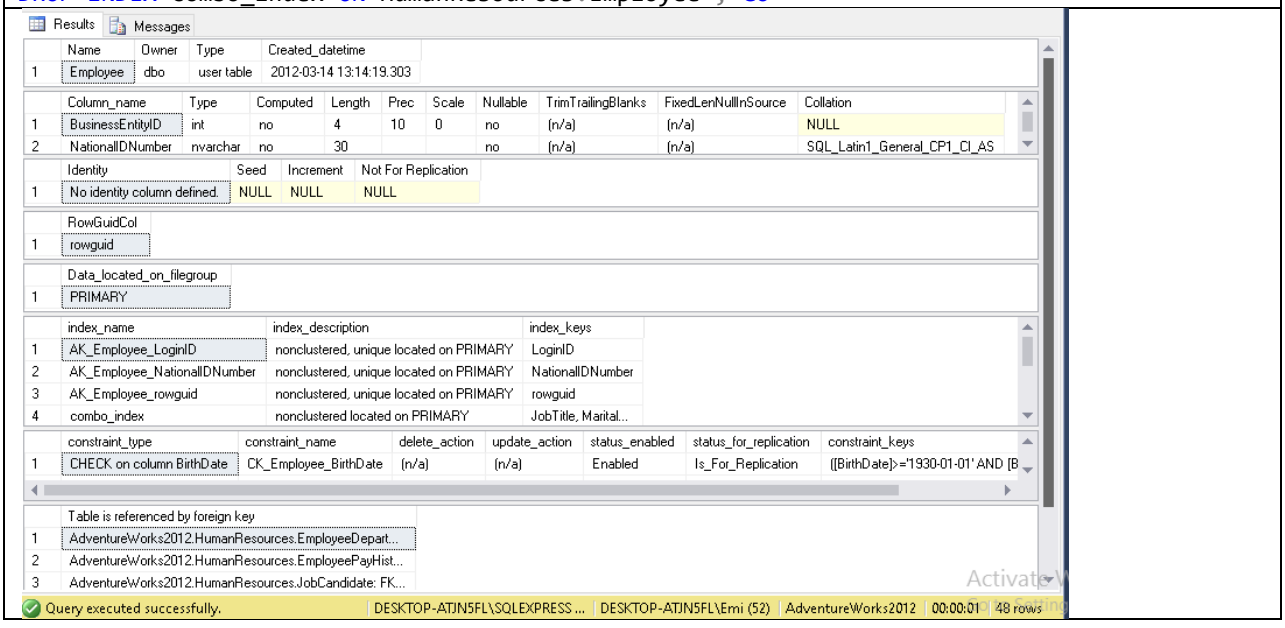



```sql
-- If an index is created with key columns that match those in the ORDER BY clause in the query, the SORT operator can be eliminated in the query plan and the query plan is more efficient.
CREATE NONCLUSTERED INDEX IX_PurchaseOrderDetail_RejectedQty
ON Purchasing.PurchaseOrderDetail (RejectedQty DESC, ProductID ASC, DueDate, OrderQty);
-- After the query is executed again, the following execution plan shows that the SORT operator has been eliminated and the newly created nonclustered index is used.
```



Index for Multiple Columns - works similarly to a sorting process on multiple columns. If an index is defined for two columns, the index key is composed by values from those two columns. It will be used to speed up the search process based on the same columns in the same order as the index definition (i.e. for an index called "combo_index" for "url" and "counts", "combo_index" will be used only when searching or sorting rows by "url" and "counts").

```sql
-- Create an index for two columns
select * from HumanResources.Employee
CREATE INDEX combo_index ON HumanResources.Employee(JobTitle, MaritalStatus); GO
-- View indexes
EXEC SP_HELP [HumanResources.Employee]; GO
-- drop index
DROP INDEX combo_index ON HumanResources.Employee ; GO
```

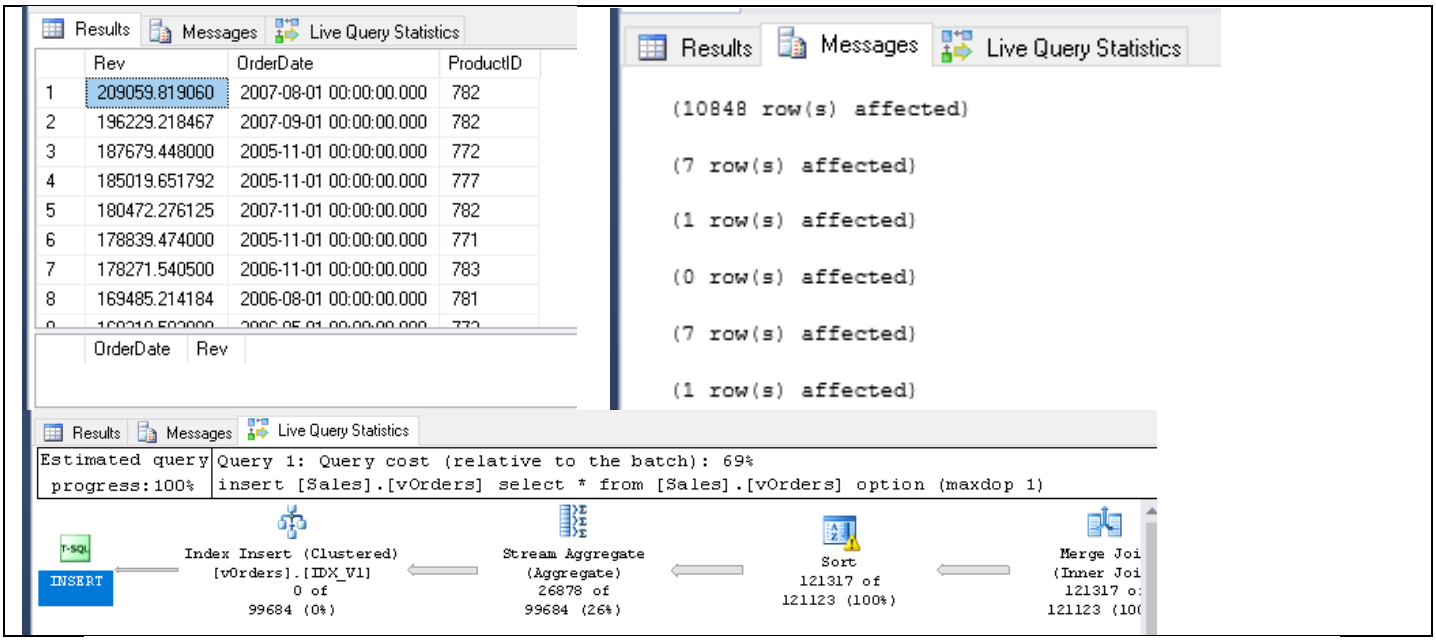


Create Indexed Views

The first index created on a view must be a unique clustered index. After the unique clustered index has been created, there can be more nonclustered indexes. Creating a unique clustered index on a view improves query performance because 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 view does not have to be referenced in the query for the optimizer to consider that view for a substitution. To create an indexed view must:

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

```
-- To create an indexed view
-- 1.  In Object Explorer, connect to an instance of Database Engine.
-- 2.  On the Standard bar, click New Query.
-- 3.  Copy and paste the following example into the query window and click Execute. The example
creates a view and an index on that view. Two queries are included that use the indexed view.
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
```



When SCHEMABINDING is specified, the base table/s cannot be modified in a way that would affect the view definition. In SCHEMABINDING, the *select_statement* must include the two-part names (*schema.object*) of tables, views, or user-defined functions that are referenced. All referenced objects must be in the same database. Views or tables that participate in a view created with the SCHEMABINDING clause cannot be dropped unless that view is dropped or changed so that it no longer has schema binding. Otherwise, the Database Engine raises an error. SCEMABINDING cannot be used in Stored Procedures. So, by including the SCHEMABINDING clause the view/function are protected from unexpected changes to the tables that underneath them.

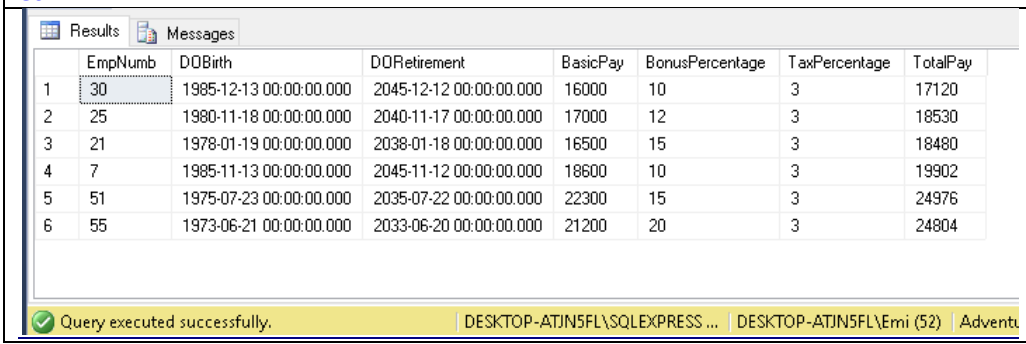
Indexes on Computed Columns

You can define indexes on computed columns as long as the following requirements are met:

- Ownership requirements - All function references in the computed column must have the same owner as the table.
- Determinism requirements – expressions are deterministic if they always return the same result for a specified set of inputs. The *computed_column_expression* must be deterministic (All functions (user-defined, built-in functions) that are referenced by the expression are deterministic and precise and also all columns that are referenced in the expression come from the table that contains the computed column; No column reference pulls data from multiple rows; The *computed_column_expression* has no system data access or user data access;). Any computed column that contains a common language runtime (CLR) expression must be deterministic and marked PERSISTED before the column can be indexed.
- Precision requirements
- Data type requirements
- SET option requirements

```
-- indexes on computed columns
-- Create UDF to use in computed column expression and create table with computed columns and insert sample data
USE AdventureWorks2012; GO
```

```sql
-- Create UDF to use in computed column expression
CREATE FUNCTION [dbo].[UDF_CalculatePay] ( @basicPay INT, @BonusPercentage TINYINT, @TaxPercentage TINYINT)
RETURNS INT
WITH SCHEMABINDING
AS
BEGIN
DECLARE @TotalPay INT
SET @TotalPay = @basicPay + @basicPay*@bonusPercentage/100 - @basicPay*@taxPercentage/100
RETURN @TotalPay
END
GO
IF OBJECT_ID('CCIndexTest', 'U') IS NOT NULL
DROP TABLE CCIndexTest
GO
-- Create table CCIndexTest with two computed columns
CREATE TABLE [dbo].[CCIndexTest](
[EmpNumb] [INT] NOT NULL,
[DOBirth] [DATETIME] NULL,
[DORetirement] AS (DATEADD(YEAR,(60),[DOBirth])-(1)) PERSISTED,
[BasicPay] [SMALLINT] NULL,
[BonusPercentage] [TINYINT] NULL,
[TaxPercentage] [TINYINT] NULL,
[TotalPay] AS [dbo].[UDF_CalculatePay] ( basicPay, BonusPercentage, TaxPercentage)
) ON [PRIMARY]
GO
-- Insert sample data
INSERT INTO dbo.CCIndexTest (empNumb, DOBirth, BasicPay, BonusPercentage, TaxPercentage)
SELECT 30 ,'1985-12-13', 16000, 10, 3 UNION ALL
SELECT 25 ,'1980-11-18', 17000, 12, 3 UNION ALL
SELECT 21 ,'1978-01-19', 16500, 15, 3 UNION ALL
SELECT 7 ,'1985-11-13', 18600, 10, 3 UNION ALL
SELECT 51 ,'1975-07-23', 22300, 15, 3 UNION ALL
SELECT 55 ,'1973-06-21', 21200, 20, 3
GO
-- Select data from dbo.CCIndexTest
SELECT * FROM dbo.CCIndexTest
GO
```



Results | Messages

	EmpNumb	DOBirth	DORetirement	BasicPay	BonusPercentage	TaxPercentage	TotalPay
1	30	1985-12-13 00:00:00.000	2045-12-12 00:00:00.000	16000	10	3	17120
2	25	1980-11-18 00:00:00.000	2040-11-17 00:00:00.000	17000	12	3	18530
3	21	1978-01-19 00:00:00.000	2038-01-18 00:00:00.000	16500	15	3	18480
4	7	1985-11-13 00:00:00.000	2045-11-12 00:00:00.000	18600	10	3	19902
5	51	1975-07-23 00:00:00.000	2035-07-22 00:00:00.000	22300	15	3	24976
6	55	1973-06-21 00:00:00.000	2033-06-20 00:00:00.000	21200	20	3	24804

Query executed successfully. DESKTOP-ATJN5FL\SQLEXPRESS ... DESKTOP-ATJN5FL\Emi (52) Adventu

So, there is a sample table with two computed columns.

```sql
-- STEP 2: Check computed columns for index creation using COLUMNPROPERTY ISINDEXABLE
SELECT
(SELECT
CASE COLUMNPROPERTY( OBJECT_ID('dbo.CCIndexTest'), 'DORetirement','IsIndexable')
WHEN 0 THEN 'No'
WHEN 1 THEN 'Yes'
END)
AS 'DORetirement is Indexable ?',
```

```
(
SELECT
CASE COLUMNPROPERTY( OBJECT_ID('dbo.CCIndexTest'),'TotalPay','IsIndexable')
WHEN 0 THEN 'No'
WHEN 1 THEN 'Yes'
END)
AS 'TotalPay is Indexable ?'
GO
```

Results | Messages

	DORetirement is Indexable ?	TotalPay is Indexable ?
1	Yes	Yes

```
-- Both of our computed columns are indeaxble.
```

References:

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?view=sql-server-2017
https://www.techonthenet.com/sql_server/indexes.php
https://technet.microsoft.com/en-us/library/jj835095(v=sql.110).aspx
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-2017
https://www.sqlshack.com/what-is-the-difference-between-clustered-and-non-clustered-indexes-in-sql-server/
https://use-the-index-luke.com/sql/where-clause/the-equals-operator/concatenated-keys
http://dba.fyicenter.com/faq/sql_server/Creating_an_Index_for_Multiple_Columns.html
https://docs.microsoft.com/en-us/sql/relational-databases/views/create-indexed-views?view=sql-server-2017
https://www.sqlshack.com/sql-server-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/indexes-on-computed-columns?view=sql-server-2017
https://www.sqlshack.com/how-to-create-indexes-on-sql-server-computed-columns/
https://www.mssqltips.com/sqlservertip/1702/how-to-create-indexes-on-computed-columns-in-sql-server/
https://www.mssqltips.com/sqlservertip/3511/sql-server-covering-index-performance/
https://docs.microsoft.com/en-us/sql/database-engine/sql-server-business-continuity-dr?view=sql-server-2017
https://www.techonthenet.com/sql_server/intersect.php
https://www.mssqltips.com/sqlservertip/4673/benefits-of-schemabinding-in-sql-server/
https://docs.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-2017
https://sqlstudies.com/2014/08/06/schemabinding-what-why/
https://ask.sqlservercentral.com/questions/2582/what-is-schemabinding.html
https://docs.microsoft.com/en-us/sql/t-sql/functions/charindex-transact-sql?view=sql-server-2017