What Is an Index?

* An index can be best described as a pointer to data in a table. An index in a database is very similar to an index in the back of a book or the phone book. Indexes are created on the table or the view

Purpose of index

* The purpose of an index is to retrieve data from a table efficiently and fast

Types of indexes

* Clustered Indexes

Clustered indexes sort the data rows in the table or view based on their key values, as such there can be only one clustered index per table. When a table does not have an index, it is referred to as a heap. When you create a clustered index, it does not require any additional disk space

* Non-clustered Indexes

A Nonclustered index have a structure separate from the data rows, much like the index in back of a book; and as such does require disk space, as it’s a separate object. A nonclustered index contains a pointer to the data row that contains the key value. For a heap, a row locator is a pointer to the row. For a clustered table, the row locator is the clustered index key.

#### Composite Indexes

|  |  |
| --- | --- |
|  | A composite index is an index on two or more columns of a table. You should consider performance when creating a composite index because the order of columns in the index has a measurable effect on data retrieval speed. Generally, the most restrictive value should be placed first for optimum performance. However, the columns that will always be specified should be placed first. |

Other less used indexes (depends)

* Covering Indexes
* Full-text
* Filtered Indexes
* Column-based Indexes
* Spatial
* XML

Creating indexes using TSQL and GUI

CREATE INDEX index\_name  
ON table\_name (column\_name)

CREATE UNIQUE INDEX index\_name  
ON table\_name (column\_name)

CREATE NONCLUSTERED INDEX [indexname]

ON table\_or\_view\_name ([columnname] ASC|DESC)

DROP INDEX index\_name ON table\_name

--CREATE CLUSTERED INDEX index\_name

--ON table\_name (column\_name)

EXAMPLE OF INDEXES AND CLUSTERED INDEX

--Insert 1000 records from RedGate app

USE [SQL2]

GO

CREATE TABLE [dbo].[PhoneBook](

[PhoneBookID] [int] NULL,

[lname] [varchar](50) NULL,

[fname] [varchar](50) NULL,

[phone] [varchar](50) NULL

) ON [PRIMARY]

GO

--Insert 1000 records from RedGate app

--View the 1000 records. Notice that the last name is not in any particular order

SELECT \* FROM PHONEBOOK

--Insert record into phonebook table and notice the row is inserted after 1000 rows, as there is no index and the Lname is in no particular order

Insert into PHONEBOOK

values (1001,'Abba','Sara','555-1212')

SELECT \* FROM PHONEBOOK

--repeat with another record

Insert into PHONEBOOK

values (1002,'Turner','Mike','805-555-1212')

SELECT \* FROM PHONEBOOK

--Create a clustered index on table phonebook and column Lname so that the last name is alphabetized (its sorted by the clustered index created)

USE [SQL2]

GO

CREATE CLUSTERED INDEX [Idx\_PhoneBook\_Lname] --<< Index name with prefix, table name and column name (convention for clarity)

ON PhoneBook(Lname ASC) --<< Table and column name

GO

--View the 1000 records. This time notice that the last name IS in order by the last!

SELECT \* FROM PHONEBOOK

--Now with the clusttered index in place, if we insert a record, it will automatically be inserted in the sorted order.

Insert into PHONEBOOK

values (1003,'Briggham','Johm','777-555-1212')

SELECT \* FROM PHONEBOOK

USE [SQL2]

GO

Drop table PHONEBOOK

NON CLUSTERED INDEX SCRIPT

--View data with clustered index (lname column is still sorted)

SELECT\*

FROM [SQL2].[dbo].[PhoneBook]

-- Create another clustered index causes an issue (on fname) because you can only have one clustered index

USE [SQL2]

GO

CREATE CLUSTERED INDEX [Idx\_PhoneBook\_phonebookid]

ON PhoneBook(phonebookid ASC)

Drop index [PhoneBook].[Idx\_PhoneBook\_lname]

--Can create an single clustered index that has multiple columns

CREATE CLUSTERED INDEX [Idx\_PhoneBook\_Fname\_Lname]

ON [dbo].[PhoneBook](lname ASC,fname ASC) --<< multiple columns

SELECT\*

FROM [SQL2].[dbo].[PhoneBook]

--Example of non clustered index

USE [SQL2]

GO

CREATE NONCLUSTERED INDEX [NC\_Ind\_PhoneBook\_ fname] --<< notice that after creation of a non clustered index, the data is not sorted

ON PhoneBook(fname ASC)

SELECT\*

FROM [SQL2].[dbo].[PhoneBook]

--Can create multiple non clustered index (On lname)

USE [SQL2]

GO

CREATE NONCLUSTERED INDEX [NC\_Ind\_PhoneBook\_phone] --<< second creation of an index and phone

ON PhoneBook(phone ASC)

SELECT\*

FROM [SQL2].[dbo].[PhoneBook]

**When should indexes be considered and**

**What factors influence creation of indexes?**

When the table is large in size (row count)

* Contains millions if not 100’s of millions rows

When the table is used frequently

* The table is used more than just infrequently

When the columns are frequently used in the WHERE clause

* The query against the database has where clause

Columns that are frequently referenced in the ORDER BY and GROUP BY clauses

* If you’re sorting data, then its beneficial

Indexes should be created on columns with a high number of unique values

* Create indexes on columns that are all virtually unique such as integer

Create a single column index on single where clause query

* If the where clause has a single column then create a single index

Create a multiple column index on multiple where clause query

* If the where clause has multiple columns then create multiple index

Build index on columns of integer type

* Integers take less space to store, which means the query will be faster

The order of the composite index matters, so place the column in the index as with the where clause

* Use the column with the lowest cardinality first, and the column with the highest cardinality last, which means keep the unique column first and less unique column last (select distinct col1 from table)

Testing is the key in determining what will work for your environment. Play with different combinations of indexes, no indexes, single-column indexes, and composite indexes.

**When Should Indexes Be Avoided?**

Indexes should not be used on small tables

* Smaller tables do better with a table scan

Indexes should not be used on columns that contain a high number of NULL values.

* Maintenance on the index can become excessive

Don’t use an index that will return a high percentage of data

* few distinct values such as gender

Creating indexes come with a cost

* Indexes take up disk space
* INSERT, UPDATE, and DELETE operations become slower, since the database system need to update the values in the table, and it also needs to update the indexes