Seminar 5

26th November 2021

# Indexes (Part I)

Index- structured stored on the disk,

* Associated with a view or table
* Used to optimize queries, delete statements, join etc

If we have great indexing => fast app

If we have poor indexing => slow down the app

General syntax:

CREATE [UNIQUE] [CLUSTERED] [NONCLUSTERED]

INDEX Index\_name

ON <object\_name\_table\_or\_view> (column [ASC| DESC], …)

[INCLUDE (col\_name , …)]

[WHERE <filter\_condition>]

[WITH (index\_option, …)]

Index characteristics:

* Clustered / non-clustered
* Unique / non-unique
* Search key – single-column / multicolumn
* Key columns / Non-key columns
* Columns in the index – ascending /descending order
* Non-clustered indexes filtered / full-table

## Clustered/Non-clustered index

|  |  |
| --- | --- |
| **Clustered** | **Non-clustered** |
| * Data rows in the table are stored on the disk, based on the index search key   CREATE CLUSTERED INDEX Index\_Name  ON Schema\_Name.TableName(Column)   * Data pages in a clustered index always include **all the columns** in the table * In sql server, all tables/views must have at most 1 clustered index * Can be used for the most frequently run queries * High degree of uniqueness * Can be used in range queries (BETWEEN) * Columns taking part in the search key: * Shouldn’t be frequently modified * Should be narrow   View as a B+ tree  Diagram  Description automatically generated  Index rows (root node):  1, 50, 100  Index rows (Intermediate level):  1, 2, 3, …  Index rows (Intermediate level):  50, 51, 52, 53, …  Leaf node for index 2:  Id=2, Name =”Person’s name” ….   * When creating a PK on a table and a clustered index is NOT already defined and there isn’t a non-clustered index specified on the PK => the PK has associated a clustered index (a clustered index is created on the fields of the PK) | * Contains key values and pointers to the data in table (heap/ clustered table)   Heap table – table without a clustered index, no particular order is followed when storing the records on the disk  CREATE INDEX Index\_Name  ON Schema\_Name.TableName(Column)  (default: non-clustered and non-unique index)   * Indexed order is independent from the order in which records are stored on the disk * SQL Server – we can have at most 999 non-clustered indexes   Non-clustered index or attribute age  (key, RID, cols\_included\_in\_the\_index)  RID – row id, row locator |

Clustered/non-clustered indexes – at most 16/32 columns, 900 bytes / 1700 bytes

## Unique indexes

* Uniqueness – property important to the query optimizer
* A unique index guarantees that the search key contains just unique values (no duplicates)
* Specify unique indexes when knowing that the values of the columns in the search key are always unique

Example:

Select \*

From Persons

Where Name = ‘Anna’

Create unique index idx\_unique\_name\_persons

ON Persons(Name)

## Key/Non-key index columns

* Key columns: columns involved in the index search key
* Non-key columns
  + Specified in the INCLUDE clause
  + CREATE INDEX Index\_name

ON TableName(KeyColumn)

INCLUDE (Col1, Col2, Col3, …)

(key, RID, Col1, Col2, Col3, …)

Example:

SELECT Name, Address

FROM Persons

WHERE Age = 20

* Covering index – contains all the columns required in a query
* Benefits:
  + Query Columns are directly accessed from the index
  + Non-key columns may include columns of data types that are not allowed to be used ars keys: varchar(max), nvarchar(max), varbinary(max)
  + When computing the size of the index, these non-key columns are not included in the index’s size

## Filtered indexes

* Optimized non-clustered indexes
* Can be used in queries that select a data subset

CREATE NONCLUSTERED INDEX idx\_ncl\_filtered

ON Orders (CustomerId, ProdId)

WHERE OrderDate IS NOT NULL

* Advantages:
  + Improve query performance
  + Reduced index:
    - Maintenance cost reduced
    - Storage cost reduced

## Index design

1. Characteristics of the database
2. Properties of the data, business behind the data
3. Most frequently used queries
4. Columns queried in selects
5. Best storage location for the index

Guidelines:

1. Database-related aspects
   1. Having too many defined indexes, may deteriorate the performance of INSERT, DELETES, UPDATES and MERGE statements (the modifications done on the table are propagated to the indexes)
   2. Small table indexing is useless, even detrimental
2. Query- related aspects
   1. Create non-clustered indexes on columns often used in WHERE or JOIN conditions
   2. Use of covering indexes may improve performance
   3. In a single statement, we are recommended to modify as many records are possible
3. Column-related aspects
   1. Length of the index key and key columns – as short as possible, especially for the clustered indexes
   2. Clustered indexes – better on unique and non-key columns
   3. Data types not used for keys: text, ntext, image, varchar(max), nvarchar(max) varbinary(max)
   4. Column uniqueness
   5. Data distribution in the columns
      1. Avoid indexes on columns with a small number of distinct values
   6. Filtered indexes – must be created on well-defined (through a condition) subsets of data
   7. Order of columns in multicolumns indexes:
      1. First positions - columns used in equality (=), columns used in inequality (<, >, BETWEEN)
      2. The rest of the columns ordered by distinctness (the cols with more distinct values should go first)
   8. Consider indexing computed columns

## Indexes for delete statements

Example: Employees and Departments

Each employee has a did FK to table Departments

When deleting a department, SQL server will retrieve data about records that have a FK to Departments

If there is an index on the FK of the Employees table than the SQL server may use it and optimize the delete

Else the SQL server needs to scan all Employees table to search for the records having the FK equal to the deleted record

As conclusions, creating indexes on FK columns can improve delete performance

SQL statements when working with indexes:

* Disable index:
  + ALTER INDEX NameIndex ON Schema\_name.TableName DISABLE
  + Table

    Description automatically generated
* Enable the indexes:
  + ALTER INDEX IndexName

ON SchemaName.TAbleName REBUILD

* sys.indexes
* sp\_help\_index tableName -> each row has info about an index stored in table tableName