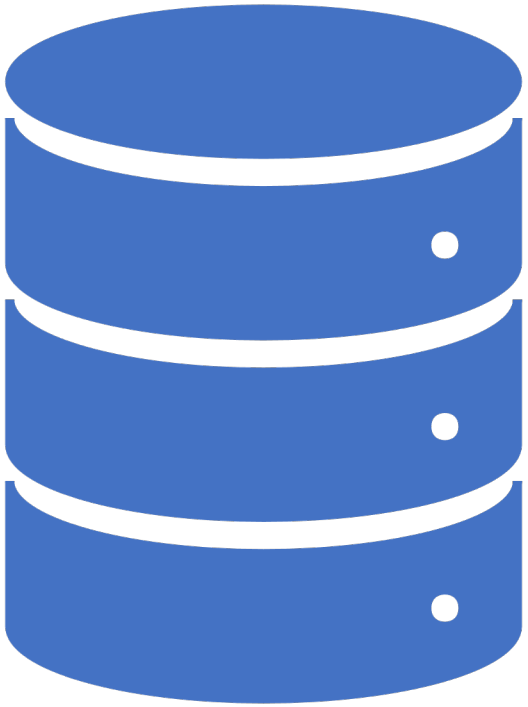


Seminar 5

Indexes
- SQL Server -

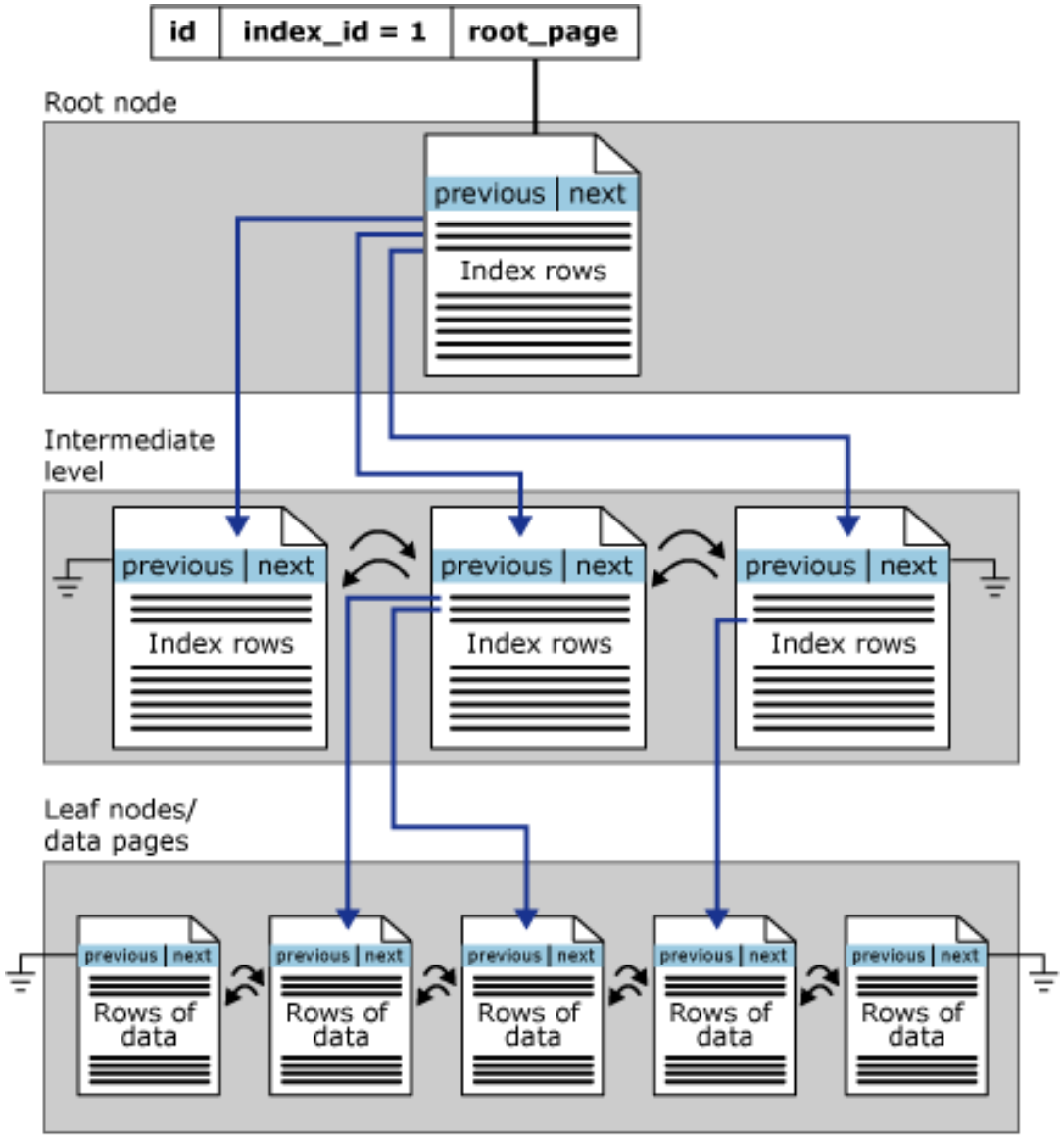
Indexes

Index = a structure stored on the disk that is associated with a table or a view and increase the speed in which the records are returned (optimizes retrieval operations on the table / view)

- A good indexing: increase the performance and the applications are faster
- A poor indexing: performance problems and can slow the DBMS
- Helps to find quicker the records that follows to be returned
- If there is no index defined, SQL Server check each record from the table to see if it contains or not the required information in the query (Table Scan)
- Table Scan = scan the entire table to find the required information
- **Indexes** are organized as the B+ tree

Indexes

- Each page in an index B+ tree: an index node.
- The top node of the B+ tree: the root node.
- The bottom nodes in the index: the leaf nodes.
- Any index levels between the root and the leaf nodes: intermediate levels.
- In a clustered index, the leaf nodes contain the data pages of the underlying table.
- The root and intermediate level nodes contain index pages holding index rows.
- Each index row contains a key value and a pointer to either an intermediate level page in the B+ tree, or a data row in the leaf level of the index.
- The pages in each level of the index are linked in a doubly-linked list.
- The pages in the data chain and the rows in them are ordered on the value of the clustered index key.
- All the inserts are performed in the moment in which the key value in the inserted row fits in the ordering sequence among existing rows.



The structure of a clustered index in a single partition

[https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver15]

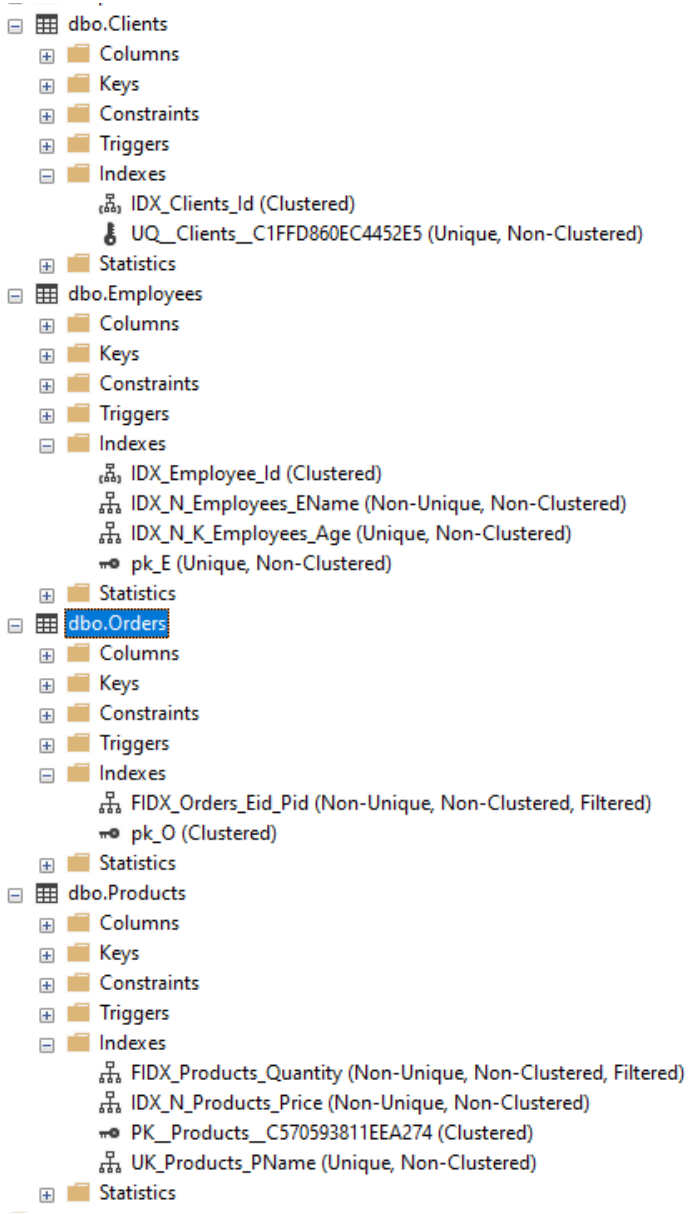
Indexes

Index syntax:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
        INDEX index_name
ON <object> ( column [ ASC | DESC ] [ ,...n ] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WHERE <filter_predicate> ]
```

Index characteristics:

- Clustered index / Non-clustered index
- Unique index / Non-unique index
- Search key: single-column index / multi-column index
- Key columns / Non-key columns
- Ascending / Descender order on the columns from the index
- Full table / Filtered for the Non-clustered index



Clustered versus Non-clustered Indexes

Clustered Index

- Defines the physical order in which the data from the table is stored (if the ***clustered index*** contains multiple columns, then the data will be stored in sequential order with respect to the columns: 1st column, 2nd column, …)
- Clustered index – automatically created when the primary key is defined (on create or alter statement)
- Only ***one clustered index*** can be defined ***per table*** (the data can be physical stored in just one order)
- The ***data pages*** from a ***clustered index*** always contain ***all the columns from the table***
- The data rows from the table are kept sorted, based on the values of the search key

```
CREATE CLUSTERED INDEX Index_Name
ON Schema_Name.Table_Name(Column(s)) [ASC | DESC]
```

Non-clustered Index

- Contains ***key values*** and ***pointers to the data*** in the table, from the ***heap / clustered index*** as part of the index key
- ***Many non-clustered indexes*** can be defined ***per table***
- SQL Server allows a maximum of ***999 non-clustered indexes per table***

```
CREATE [NONCLUSTERED] INDEX Index_Name
ON Schema_Name.Table_Name(Column(s)) [ASC | DESC]
```

Clustered versus Non-clustered Indexes

- An ***index key*** (clustered / non-clustered index) can contain ***at most 16 columns*** and ***900 bytes***

Clustered versus Non-clustered Index

- When a primary key is defined on a table, if there is no clustered index defined on it and any non-clustered index specified, ***an unique clustered index*** is defined ***on the fields of the primary key***
- When a primary key is created on a table and also a clustered index is defined, then ***a non-clustered unique index*** will be created
- If all the columns returned by a query are in an index, the index is called ***covering index*** and the query is called ***covered query***

Clustered index

- Should be used on the ***queries*** that are ***executed frequently***
- Can be used in ***range queries***
- Has a ***high degree of uniqueness***
- The columns that are part of the ***search key***: should be narrow and shouldn't be frequently changed
- It is not good to define a ***clustered index*** on the columns that are frequently modified, because SQL Server should also modify constantly the physical order of the data

Clustered versus Non-clustered Index

Clustered Index

- Define an unique clustered index:

  CREATE UNIQUE CLUSTERED INDEX Index_Name
  ON Schema_Name.Table_Name(Column(s)) [ASC | DESC]

Non-clustered Index

- When a ***UNIQUE constraint*** is defined on a table, an ***unique non-clustered index*** is created on the column or columns on which the UNIQUE constraint is defined
- Define a non-clustered and non unique index:

  CREATE UNIQUE [NONCLUSTERED] INDEX Index_Name
  ON Schema_Name.Table_Name(Column(s)) [ASC | DESC]

e.g. CREATE NONCLUSTERED INDEX IDX_Students_Name_ASC
ON Students(Name ASC);

or

CREATE INDEX IDX_ Students_Name_ASC
ON Students(Name);

Unique index

- Guarantees that the ***search key*** contains ***no duplicate values***
- It makes sense only when there are no entries with identical values in the key columns
- Assure the uniqueness, an useful information for the query optimizer

e.g. CREATE UNIQUE CLUSTERED INDEX IDX_Studenys_Sid_ASC
ON Students (Sid ASC);

Unique index versus non-unique index

- An ***unique index*** defined on one or multiple columns assure the ***unicity of the values*** on the level of the ***column*** or ***combination of columns***
- Examples:
  - if an unique non-clustered index is created on the *Students* table on the column *Name*, there cannot be 2 records with the same *Name* value in the table
  - if an unique non-clustered index is created on the *Students* table on the columns *Name* and *Surname*, there cannot be 2 records with the same values for the *Name* and *Surname* in the table
  - if an unique non-clustered index is created after inserting values in the table and there are duplicates on the column or columns for which the unique non-clustered index was defined, the operation of creating the index, fails

Unique index

Unique index versus non-unique index

- To ***create an unique index*** on a table, first ***should be eliminated the duplicate values*** from the column or columns for which the index will be defined
- An ***unique index*** guarantees that ***each value*** (including NULL) ***for*** the ***column*** in which was ***defined appears only once in the table***

e.g. unique non-clustered index on a single column

```
CREATE UNIQUE INDEX IDX_Courses_Title_DESC_UQ
ON Courses (Title DESC);
```

e.g. unique non-clustered index on multiple columns

```
CREATE UNIQUE INDEX IDX_Students_Name_ASC_Surname_ASC_UQ
ON Students (Name ASC, Surname ASC);
```

Unique index

Unique index versus non-unique index

- For the ***unique index*** can be set the option **IGNORE_DUP_KEY**
- If **IGNORE_DUP_KEY = ON** for a statement INSERT, then will be inserted all the records that doesn't contain duplicated values, and the records that contain duplicate values will be ignored and won't be inserted in the table

e.g. CREATE UNIQUE INDEX IDX_Students_Name_ASC_UQ
ON Students (Name ASC)
WITH (IGNORE_DUP_KEY=ON);

- The ***non-unique index*** are not unique and allows the adding / insert of duplicate values in table

Key versus Non-key Index columns

Key columns

- Columns specified on the create index statement
- Columns in the search key

Non-key columns

- Columns specified in the **INCLUDE** clause when creating a non-clustered index

```
CREATE INDEX Index_Name
 ON Schema_Name.Table_Name(Column(s)) [ASC | DESC]
 INCLUDE (nonkey_column_name(s))
```

e.g.
```
CREATE INDEX IDX_Students_Name_ASC_Surname_ASC
    ON Students (Name ASC, Surname ASC)
    INCLUDE (Dob, Email);
```

Key versus Non-key Index columns

Advantages of non-key columns

- Columns can be accessed with an ***Index Scan*** from the index
- The ***data types*** that are ***not allowed in key columns*** can be ***used in non-key columns*** (varchar(max), nvarchar(max), varbinary(max), image, text, …)
- Computed columns can be included if their values are deterministic
- The columns that appear in the **INCLUDE** clause (non-key columns) doesn't count in those 900 bytes limit of a key index, imposed by SQL Server

Single colum versus Multi-column index

Single column index

- An index defined on a single column (that contain only one key column in the key index)

Multi-column index

- An index defined on multiple columns (that contain multiple key columns in the key index)

- If the index is also used to sort / order the records that are returned, should be taken into account the ascending or descending order of the columns that are part of the key index
- For a ***single column index*** the specified order for the key column is not very important because the index can be used to sort after that column in ascending or descender order

e.g. non-unique non-clustered index on a single column

```
CREATE INDEX IDX_Students_Name_DESC ON Students(Name DESC);
```

used in the following 2 queries

```
SELECT Name FROM Students ORDER BY Name ASC;
SELECT Name FROM Students ORDER BY Name DESC;
```

Single colum versus Multi-column index

- For the ***multi-column index*** the order of the key columns is important when the index will be used to sort the records after the columns that are part of the key index

e.g. non-unique non-clustered multi-column index

```
CREATE INDEX IDX_Students_Name_ASC_Gid_ASC ON Students(Name ASC, Gid ASC);
```

used in **sorting** the following queries

```
SELECT Name, Gid FROM Students ORDER BY Name ASC, Gid ASC;
SELECT Name, Gid FROM Students ORDER BY Name DESC, Gid DESC;
SELECT Name, Gid FROM Students ORDER BY Name ASC;
SELECT Name, Gid FROM Students ORDER BY Name DESC;
```

but **cannot** be used to **sort** the following queries

```
SELECT Name, Gid FROM Students ORDER BY Name ASC, Gid DESC;
SELECT Name, Gid FROM Students ORDER BY Name DESC, Gid ASC;
SELECT Name, Gid FROM Students ORDER BY Gid ASC;
SELECT Name, Gid FROM Students ORDER BY Gid DESC;
```

Full-table versus Filtered non-clustered index

Non-clustered full-table index

- Contains all the values of the column or columns on which had been defined

Non-clustered filtered index

- Contains only the values of the column or columns for which the execution of the specified condition on the create index statement was true
- Is an optimized non-clustered index
- Can be used by queries that select from a certain subset of data
- Is better for the query performance
- The reduced index has impact on maintenance cost and storage cost

e.g. CREATE NONCLUSTERED INDEX IDX_Sid_CoId
ON Exams(Sid, CoId)
WHERE Grade IS NOT NULL

Full-table versus Filtered non-clustered index

e.g. non-unique non-clustered single column filtered index

```
CREATE INDEX IDX_Students_Name_ASC_filtered
ON Students(Name ASC)
WHERE Name > ‘D’;
```

used for the following queries

```
SELECT Name FROM Students WHERE Name > ‘D’;
SELECT Name FROM Students WHERE Name > ‘G’;
```

but **cannot** be used in the following queries

```
SELECT Name FROM Students ORDER BY Name ASC, Gid DESC;
SELECT Name FROM Students;
```

Indexes for DELETE

- The indexes can be useful also when delete data from the database, not only when data is retrieved
- When a delete operation is performed
  - SQL Server check all the tables that depend on the table from which the delete will be performed, to determine if there are dependent records to the ones that will be deleted
  - By searching for those dependent rows are examined all the foreign keys (when a record r is deleted, the systems checks whether r is referenced by other records)
    - if there is an index defined on a foreign key, SQL Server uses it to check the existence of related data
    - if there isn't an index, the system has to scan the identified table
  - So, an index defined on a ***foreign key*** could be used to find the depending records ***faster*** than normal
  - In contrary, SQL Server will check all the records from the involved table and consequently the delete operation will be slower
- The performance of the delete operations can be improved by creating indexes on the ***foreign keys***

Modify and Delete an index

- To delete an index or to modify the structure of an index (e.g. adding new column(s) in the index) – it should be ***deleted*** and after that ***created*** again

- Deactivate an index / set options

  ALTER INDEX *Index_Name*
  ***ON Schema_Name.Table_Name(Column(s)) [ASC | DESC]* DISABLE**

  e.g. ALTER INDEX IDX_Students_Name_DESC ON Students DISABLE;

- Reactivate a deactivated index

  ALTER INDEX *Index_Name*
  ***ON Schema_Name.Table_Name(Column(s)) [ASC | DESC]* REBUILD**

  e.g. ALTER INDEX IDX_Students_Name_DESC ON Students REBUILD;

- When an index become un-useful should be eliminated

  DROP INDEX index_name ON table_name;

  e.g. DROP INDEX IDX_Students_Name_DESC ON Students;

Index Design and Guidance

- Some characteristics should be analyzed to obtain the best indexes:
  - The database: Online Transaction Processing (OLTP) versus Online Analytical Processing (OLAP)
  - The most frequently executed queries
  - The columns used in the queries
  - The optimal store location for an index

- Guidelines to:
  - Database
    - Should be taken into account the presence of many indexes on a table because can deteriorate the performance of INSERT, UPDATE, DELETE and MERGE statements
    - Indexing small tables is often useless
  - Query
    - The non-clustered indexes should be created on columns that are often used in WHERE and JOIN clauses
    - The covering indexes can significantly improve the performance of queries
    - As many records as possible should be changed in a single statement

Index Design and Guidance

- Guidelines to:
  - Columns
    - The length of the key index should be as short as possible for the clustered indexes
    - Column uniqueness is important
    - Clustered indexes are better on unique / non-null columns
    - Types ntext, text, image, varchar(max), nvarchar(max), varbinary(max) cannot be used for search key fields
    - For the data distribution in the column should be avoided the indexes on the columns with a small number of distinct values; should be used filtered indexes
    - The order of columns in multicolumn indexes should be
      - first positions - columns in equality (=), inequality (>, <, BETWEEN) conditions
      - then, the rest of the columns should be ordered by distinctness
    - Consider indexing computed columns

Index Guidance

- Indexes are useful to increase the performance of the read operations, but decrease the performance of the writing operations

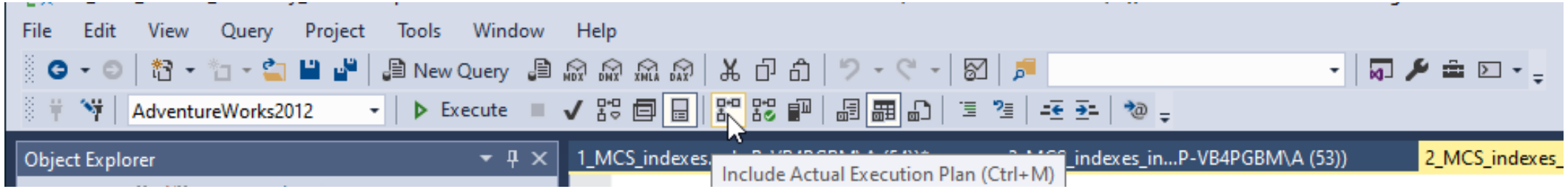
- The ***columns*** that are recommended to have ***key index*** on them are:
  - The columns that have a ***foreign key constraint*** defined on them
  - The columns that appear in the ***WHERE*** clause of the queries
  - The columns that appear in the ***ORDER BY*** clause of the queries
  - The columns that cause the ***JOIN's***
  - The columns that appear in the ***GROUP BY*** clause of the queries
  - The columns that have a ***big variety of values***

- More than the ***columns*** on which are created ***key indexes*** should be taken into consideration the ***list of fields from SELECT*** (these ones also influence the index that is used)

Index Guidance

Choose a good index

1. Execute the SELECT statement without having extra indexes especially created + check the Execution Plan
2. Create index on the column(s) involved in the SELECT statement (and its clauses)
3. Execute the SELECT statement after the index has been added on the column(s) involved + check the Execution Plan
   - If the Execution Plan is better than ***keep the index***, otherwise ***remove the index***
   - Insert extra records – to be sure the performance can be or cannot be improved



or