Fall
2025

Clustered & Non-Clustered Indexes

CS340 | TUTORIAL 10

ALI KHAWAJA

Database Files & Filegroups

SQL Server Data files contain data and objects such as tables, indexes, stored procedures, and views.

Data files can be grouped together in filegroups for allocation and administration purposes. Filegroups work as a logical container for the data files, and a filegroup can have multiple data files.

Data files can be of two types:

- Primary: The primary data file (**mdf** extension) contains startup information for the database and points to other files in the database. User data and objects can be stored in this file, and every database has one primary data file.
- Secondary: Secondary data (.**ndf** extension) files are optional and can be used to spread data across multiple files/disks by putting each file on a different disk drive.

Creating Database table on a separate data file & separate disk

To create a table on a secondary data file (and on a separate disk), we must first create a new filegroup and a physical data file (.ndf) to store the table in a secondary location.

```
USE LUMS;
GO

-- 1. Create a New Filegroup
ALTER DATABASE LUMS
ADD FILEGROUP SecondaryFG;
GO

-- 2. Add a Secondary Data File (.ndf) to the New Filegroup
ALTER DATABASE LUMS
ADD FILE (
    NAME = SecondaryDataFile,
    FILENAME = '/var/opt/mssql/data/SecondaryDataFile.ndf',
    SIZE = 5MB,
    MAXSIZE = 100MB,
    FILEGROWTH = 5MB
)TO FILEGROUP SecondaryFG;
GO
```

Part 1: Primary Key Automatically Creates a Clustered Index

By default, when you define a **PRIMARY KEY** constraint on a table and **do not** specify an index type (like NONCLUSTERED), SQL Server automatically creates a **Unique Clustered Index** on that column(s).

1. Create the Table with a Primary Key

SQL

```
USE LUMS;
GO

-- Create a table with a Primary Key
CREATE TABLE cs340.Students (
    EmployeeID INT NOT NULL PRIMARY KEY, -- This will implicitly create a
CLUSTERED index
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    DateOfBirth DATE
);
GO
```

2. Verify the Clustered Index Creation

You can use the sys.indexes system view to verify the index type created for the primary key constraint.

```
-- Check the indexes on the table
EXEC sp_helpindex 'cs340.Students';
```

Observation: The result will show an index as Clustered, Unique and Primary Key, confirming SQL Server created a clustered index automatically.

Part 2: Create Clustered Index on a Unique Column in a Secondary Data File

You can explicitly control the table's physical storage by creating the **Clustered Index** on a specified **Filegroup**. This also shows how a Clustered Index can be placed on a column other than the primary key, as long as you make the primary key NONCLUSTERED.

1. Create the Table with a Non-Clustered Primary Key

Since a table can only have **one** clustered index, you must explicitly define the **PRIMARY KEY** as **NONCLUSTERED** if you plan to create a clustered index on a different column.

```
-- Create a table with a NONCLUSTERED Primary Key
CREATE TABLE StudentAddress (
    StudentID INT NOT NULL,
    AddressLine1 NVARCHAR(100),
    AddressLine2 NVARCHAR(100),
    City NVARCHAR(50),
    PostalCode NVARCHAR(20),
    CONSTRAINT PK_StudentAddress PRIMARY KEY NONCLUSTERED (StudentID)
);
GO

---- Check the indexes on the table
EXEC sp_helpindex 'cs340.Students';
```

2. Create a Clustered Index on a Unique Column

Now, create a **UNIQUE CLUSTERED INDEX** on the PostalCode column and explicitly specify the SecondaryFG filegroup created earlier. This will MOVE the table to the files in the secondary filegroup, which could be sitting on a separate disk.

```
-- Create a Unique Clustered Index on the PostalCode column
-- and place it on the secondary filegroup.
CREATE UNIQUE CLUSTERED INDEX IX_StudentAddress_PostalCode_Clustered
ON cs340.StudentAddress(PostalCode)
ON SecondaryFG;
GO

-- Check the indexes on the table
EXEC sp_helpindex 'cs340.Students';
```

3. Verify the Index Location and Type

Use the system views to confirm that the Clustered Index is on the specified filegroup. While the previously mentioned system helper stored procedure gives you all the required info, you can write the below query to get the information yourself as well:

```
-- Check the indexes and their filegroups
SELECT
    t.name AS TableName,
    i.name AS IndexName,
    i.type_desc AS IndexType,
    f.name AS FileGroupName
FROM
    sys.indexes AS i
INNER JOIN
    sys.tables AS t ON i.object_id = t.object_id
INNER JOIN
    sys.filegroups AS f ON i.data_space_id = f.data_space_id
WHERE
    t.name = 'StudentAddress'
    AND i.index_id > 0;
```

Observation: Verify the result shows that while PK was non-clustered, the Unique clustered key forced the table to be moved to secondary filegroup.

Challenge:

1) Convert the above method to a stored procedure that accepts the table name as a parameter.
2) Create stored procedures to add dummy data into your table
3) Insert studentId using sql server sequence.
4) Use Truncate Table command and verify index stats.