



# Query Plans & Covering Index

© Copyright Microsoft Corporation. All rights reserved.

Ali Khawaja | CS340 | LUMS SBASSE



# Agenda

---

- Index Seek vs. Index Scan
- Query Statistics & IO Plans
- SARGable Queries
- Covering Index

# Index Seek vs. Index Scan



## Index Seek vs. Index Scan

- **Index Scan**
  - An Index Scan occurs when SQL Server reads the entire index to find the matching records.
  - SQL Server it loads the object (Index, Table, etc.) which it wants to read from disk into memory, then reads through that object from top to bottom looking for the records that it needs.
- An **Index Seek** is where SQL Server uses the B-tree structure of the index to directly navigate to the matching records.
  - It only needs to load specific pages into the memory to look for the data (which could already be cached in memory)

# IO Statistics & Query Plans



# IO Statistics & Query Plans

## **SET STATISTICS IO { ON | OFF }**

Causes SQL Server to display information about the amount of physical and logical IO activity generated by Transact-SQL statements.

```
USE AdventureWorks2022;
GO

SET STATISTICS IO ON;
GO

SELECT *
    FROM Production.ProductCostHistory
 WHERE StandardCost < 500.00;
GO

SET STATISTICS IO OFF;
GO
```

***Look under the messages tab in the vscode output pane to see the statistics***

# IO Statistics & Query Plans

## **SET SHOWPLAN\_ALL**

Causes Microsoft SQL Server not to execute Transact-SQL statements.

Instead, SQL Server returns detailed information about how the statements **would be executed** (a query plan)

## **SET SHOWPLAN\_TEXT**

Causes Microsoft SQL Server not to execute Transact-SQL statements.

Instead, SQL Server returns detailed information about how the statements are executed.

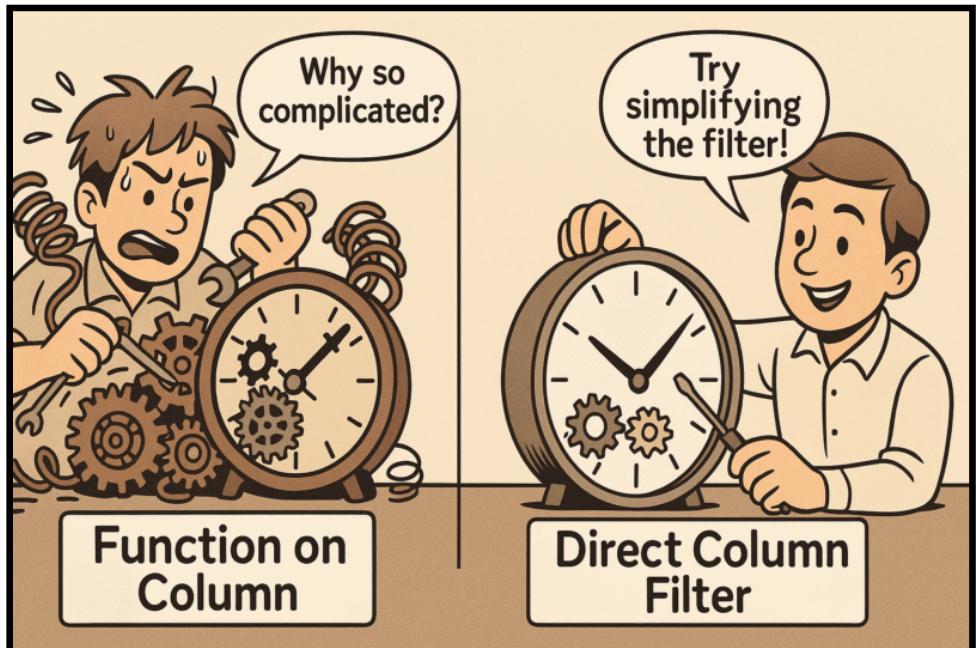
***Look beside the execute button in vscode to display the query plan***

# Search **Argument** Able Queries



## Search ARGument ABLE Queries (SARGable)

- SARGable queries enable the optimizer to determine when to use index seek and index scans
- Non-SARGable queries usually force the database to perform full index/table scans, as they can't utilize indexes due to the search criteria



## Non-SARGable Query

```
SELECT *
FROM Employees
WHERE DATEDIFF(DAY, HireDate, GETDATE()) = 1000;
```

Function **DATEDIFF(DAY, HireDate, GETDATE())** disables index seek.  
SQL Server can't jump directly to matching rows—it has to scan the entire index or table.

## SARGable Query

```
SELECT *
FROM Employees
WHERE HireDate =
    CAST(DATEADD(DAY, -1000, GETDATE()) AS DATE);
```

Now, we'll get the index seek as desired, instead of index scan

Applying functions directly to the WHERE clause blocks the use of Index

## Common Non-SARGable Patterns

- WHERE MONTH(HireDate) = 1
- WHERE HireDate  $\geq$  '2022-01-01' AND HireDate < '2022-02-01'
- WHERE DATEDIFF(day, HireDate, GETDATE()) > 365
- WHERE HireDate < DATEADD(day, -365, GETDATE())
- WHERE ABS(Salary) > 5000
- Preprocess absolute values or refactor logic
- WHERE ISNULL(DepartmentID, 0) = 5
- Use defaults during data insert or handle NULLs before filtering

# Covering Index



## What is a Covering Index?

A covering index in SQL Server is a non-clustered index that contains all the columns needed to satisfy a query

SQL Server can retrieve the data entirely from the index without having to access the base table

It's called "covering index" because it "covers" all the columns in the query:

- All columns in the SELECT, WHERE, JOIN, and ORDER BY clauses are present in the index.

## Covering Index: Index with Included Columns

```
SELECT FirstName, LastName  
FROM Employees  
WHERE DepartmentID = 5;
```

*For the above query, following index with included columns “covers” all columns required for the resultset:*

```
CREATE NONCLUSTERED INDEX IX_Employees_Department  
ON Employees (DepartmentID)  
INCLUDE (FirstName, LastName);
```

## Benefits (& Cost)

An index with non-key columns can significantly improve query performance when all columns in the query are included in the index either as key or non-key columns.

Performance gains are achieved because the query optimizer can locate all the column values within the index.

***\*\* Covering indexes can increase **storage** and **maintenance** overhead. \*\****