Microsoft

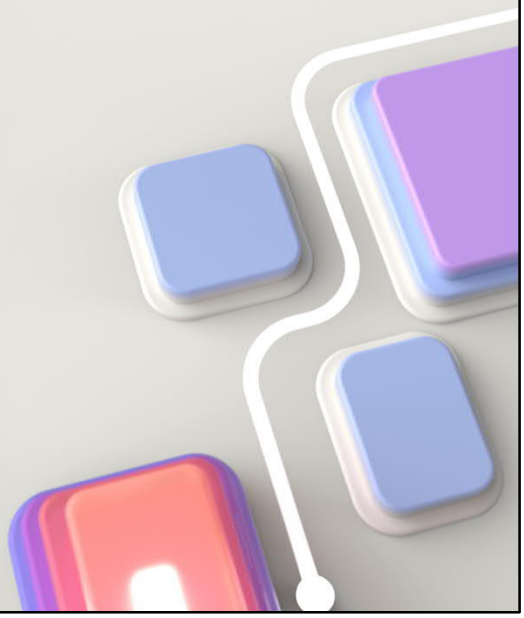
Query Plans & Covering Index

Ali Khawaja | CS340 | LUMS SBASSE

Agenda

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

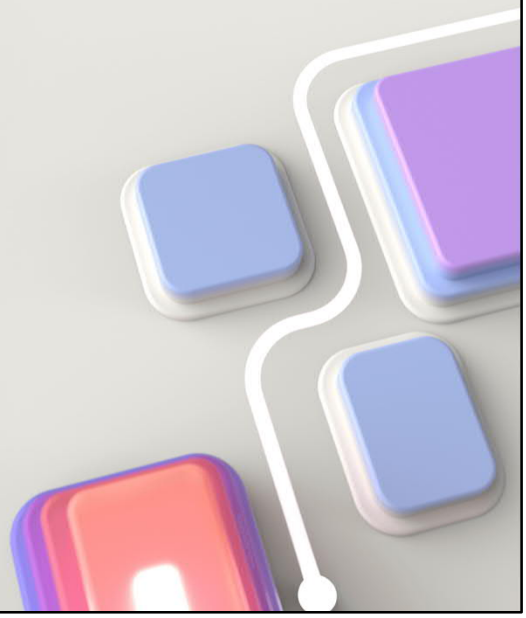
Index Seek vs. Index Scan



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

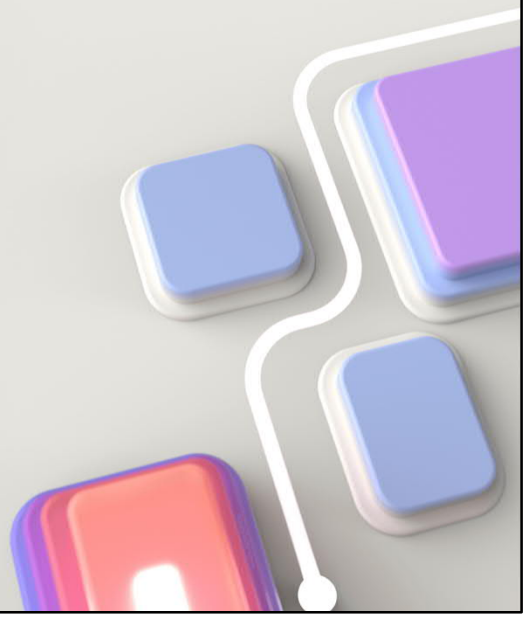
Causes Microsoft SQL Server not to execute Transact-SQL statements.

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

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

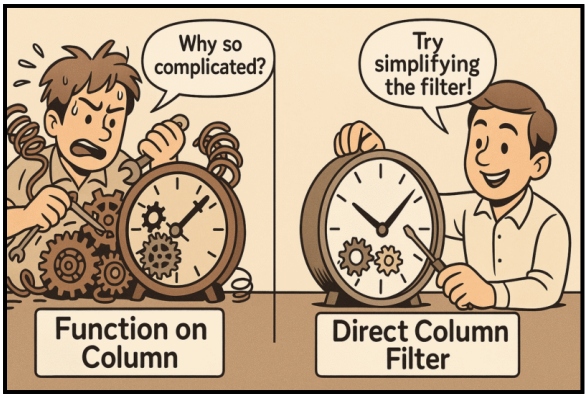
https://learn.microsoft.com/en-us/sql/t-sql/statements/set-showplan-all-transact-sql?view=sql-server-ver17
https://learn.microsoft.com/en-us/sql/t-sql/statements/set-showplan-text-transact-sql?view=sql-server-ver17

Search **Arg**ument 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



https://blog.sqlauthority.com/2025/04/23/sql-server-catching-non-sargable-queries-in-action/

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.

https://blog.sqlauthority.com/2025/04/23/sql-server-catching-non-sargable-queries-in-action/

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

https://blog.sqlauthority.com/2025/04/23/sql-server-catching-non-sargable-queries-in-action/

Common Non-SARGable Patterns

- `WHERE MONTH(HireDate) = 1`
- `WHERE HireDate ≥ '2022-01-01' AND HireDate < '2022-02-01'`

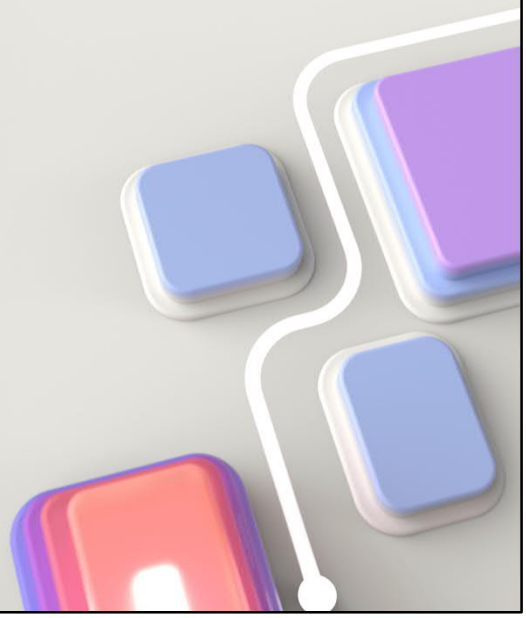
- `WHERE DATEDIFF(day, HireDate, GETDATE()) > 365`
- `WHERE HireDate < DATEADD(day, -365, GETDATE())`

- `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

https://blog.sqlauthority.com/2025/04/23/sql-server-catching-non-sargable-queries-in-action/

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

Its 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.

https://learn.microsoft.com/en-us/sql/relational-databases/indexes/create-indexes-with-included-columns?view=sql-server-ver17

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**. ***

https://learn.microsoft.com/en-us/sql/relational-databases/indexes/create-indexes-with-included-columns

Design Recommendations

- Redesign nonclustered indexes that have a large index key size so that only columns used for searching and lookups are key columns. Make all other columns that cover the query into nonkey columns. In this way, you'll have all columns needed to cover the query, but the index key itself is small and efficient.
- Include nonkey columns in a nonclustered index to avoid exceeding the current index size limitations of a maximum of 32 key columns and a maximum index key size of 1,700 bytes (16 key columns and 900 bytes prior to SQL Server 2016 (13.x)). The Database Engine doesn't consider nonkey columns when calculating the number of index key columns or index key size.
- The order of nonkey columns in the index definition doesn't affect the performance of queries that use the index.
- Avoid very wide nonclustered indexes where the included columns don't represent a narrow enough subset of the underlying table columns. If adding wide indexes, always verify if the cost of updating one extra wide index offsets

the cost of reading directly from the table.

Limitations

- Nonkey columns can only be defined on nonclustered indexes.
- All data types except text, ntext, and image can be used as nonkey columns.
- Computed columns that are deterministic and either precise or imprecise can be nonkey columns. For more information, see Indexes on computed columns.
- Computed columns derived from image, ntext, and text data types can be nonkey columns as long as the computed column data type is allowed as a nonkey index column.
- Nonkey columns can't be dropped from a table unless that table's index is dropped first.
- Nonkey columns can't be changed, except to do the following:
    - Change the nullability of the column from NOT NULL to NULL.
    - Increase the length of varchar, nvarchar, or varbinary columns.