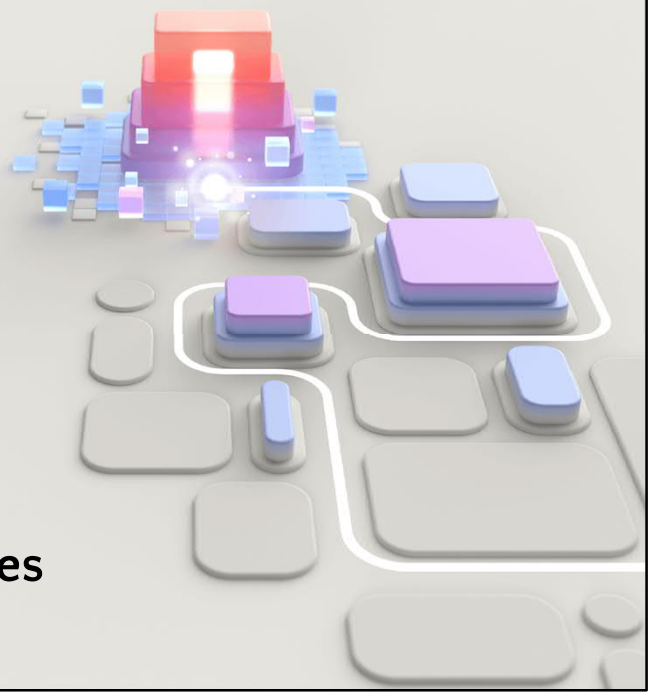
Microsoft

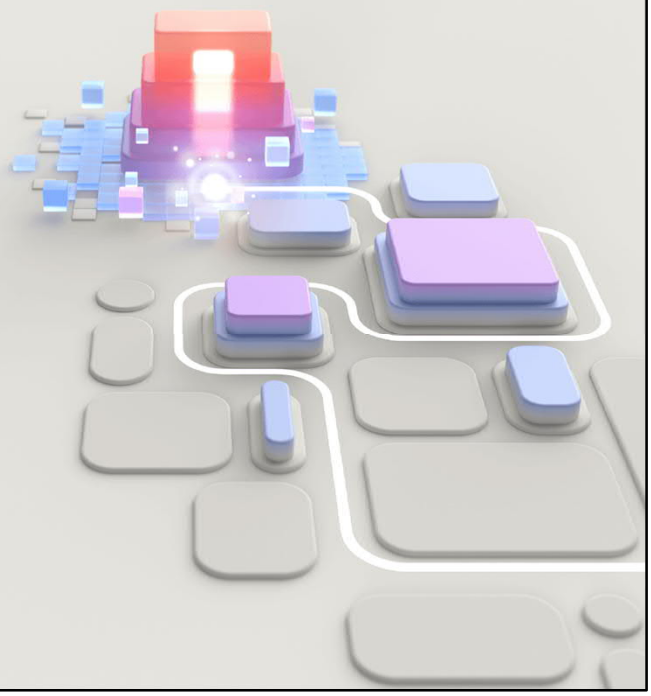


Non-Clustered Index Types

Agenda

- Unique Index
- Filter Index
- Index with Computed Columns

Microsoft



Unique Index

Unique Index

- Ensures **uniqueness** of column values in a table (data integrity).
- Useful for columns that **must not contain duplicates**,
  - **Example**: NationalIDNumber , Email
- Can be **single-column or multi-column** (combinations must be unique)
  - **Example of multicolumn**: LastName and DepartmentID
- Can be **clustered or non-clustered**.
- Helps **query optimizer** by providing additional info for efficient execution plans.

Guarantees no duplicate values → ensures data integrity.
Can be single-column or multi-column (combination unique).
Can be clustered or nonclustered.
Automatically created with PRIMARY KEY or UNIQUE constraint.
Useful for query optimizer → can produce faster execution plans.
If duplicates already exist, unique index cannot be created.
Non-key included columns can be added to cover more queries without violating uniqueness.

Usage

EmployeeID	Name	Dept	NationalID
101	Ali	IT	NID123
102	Sarah	CS	NID124
103	Omer	Physics	NID125
104	Ben	Math	NID126

Creating Index:

```
CREATE UNIQUE INDEX UX_Employees_NID
ON Employees(NationalIDNumber);
```

Query

```
SELECT * FROM Employees WHERE
NationalIDNumber = 'NID124';
```

SQL Server searches the **unique index** → finds exact row → retrieves data quickly.

How it works:

- On Insert, SQL Server checks leaf nodes of the unique index.
- If NationalID exists, insertion fails (error).
- If not: insertion succeeds, row added in B+ tree leaf node.

Unique Constraint vs Unique Index

Unique Constraints:

- No concept of filtered or covering index. Only applied to the column
- Tied to table definition
- ALTER TABLE …

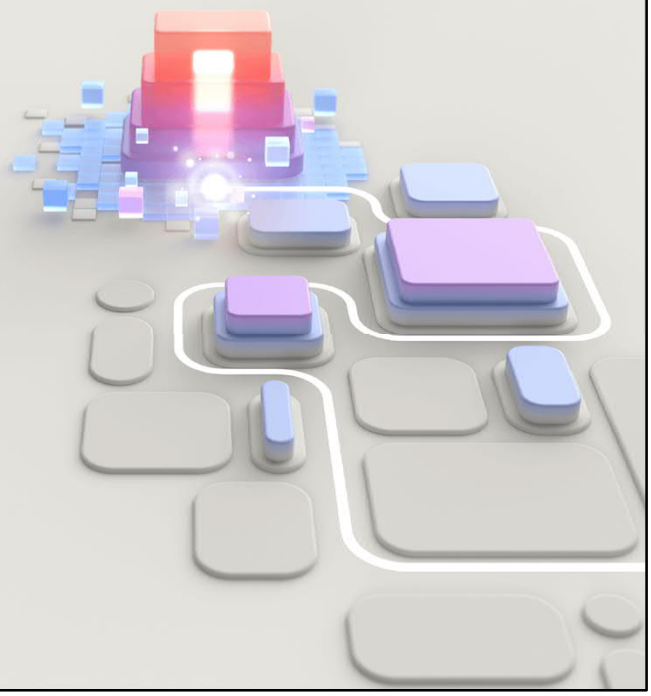
Unique Index:

- Supports **filtered indexes** and **included columns**
- **Can be included or dropped independent of the table**
- DROP INDEX <index_name> ON <db.schema.tablename>;

Question

What should happen if Unique constraint is applied to an existing column with non-unique values?

Microsoft



Filtered Index

Filtered Index

Non-clustered index that is created on a subset of rows in a table, defined by a filter condition (WHERE clause).

- **Subset of data** → Index only rows you care about
  - Example: WHERE **EndDate** IS NOT NULL
- **Sparse columns** → Great for columns with mostly NULL values.
- **Heterogeneous categories** → When table has categories (e.g., Product types), create index only for specific category.
- **Range queries** → Dates, prices, amounts where only a certain range matters.
  - Example: WHERE ID > 2 AND ID < 5

- **Covers only subset** → Queries outside filter can't use the index.
- **Maintenance overhead** if filter covers large % of table.
- **Too many small filtered indexes** → Can increase management complexity.

Filtered Index – Architecture

- It's a **non-clustered index** → built on **only rows that meet condition**.
- Has **key columns + optional included columns** (just like normal non-clustered index).
- Automatically stores **clustered key (or RID in heap)** as row locator.
- Storage is smaller → because it ignores unneeded rows.

Non-clustered + built on only rows that meet condition — *Why:* because the filter keeps the index entries to only relevant rows, making searches for that subset much smaller and faster.

Has key columns + optional included columns — *Why:* key columns let the index find rows; included columns let the index *cover* queries so SQL can return results without touching the base table.

Automatically stores clustered key (or RID in heap) as row locator — *Why:* the index needs a pointer to fetch the full row; clustered tables use the clustered key, heaps use the physical RID, so lookups are deterministic.

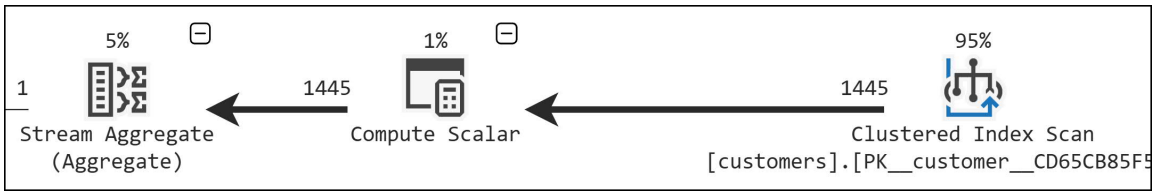
Storage is smaller (ignores unneeded rows) — *Why:* fewer indexed rows + optional includes = fewer pages on disk → less IO and lower storage cost.

Example: Filtered Indexes (cont…)

```
SELECT
    SUM(CASE
            WHEN phone IS NULL
               THEN 1
               ELSE 0
          END) AS [Has Phone],
    SUM(CASE
            WHEN phone IS NULL
               THEN 0
               ELSE 1
          END) AS [No Phone]
FROM
    sales.customers;
```

https://www.sqlservertutorial.net/sql-server-indexes/sql-server-filtered-indexes/

Example: Filtered Indexes (cont…)



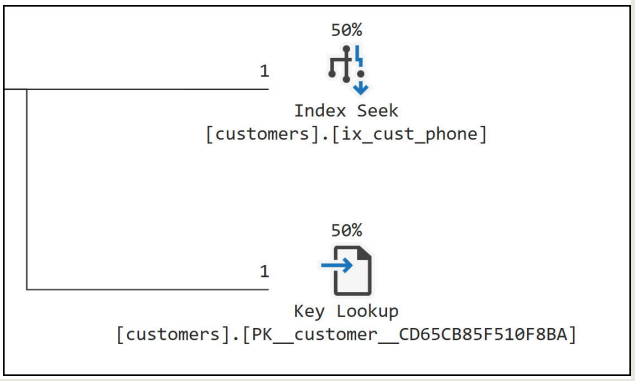
Example: Filtered Indexes (cont…)

```
CREATE INDEX ix_cust_phone
 ON sales.customers(phone)
WHERE
 phone IS NOT NULL;
```

```
SELECT
    first_name,
    last_name,
    phone
FROM
    sales.customers
WHERE phone = '(281) 363-3309';
```

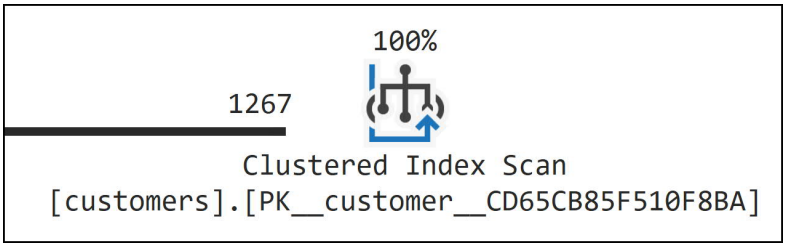
Query Plan!

```
SELECT
    first_name,
    last_name,
    phone
FROM
    sales.customers
WHERE
  phone = '(281) 363-3309';
```



Updated Query

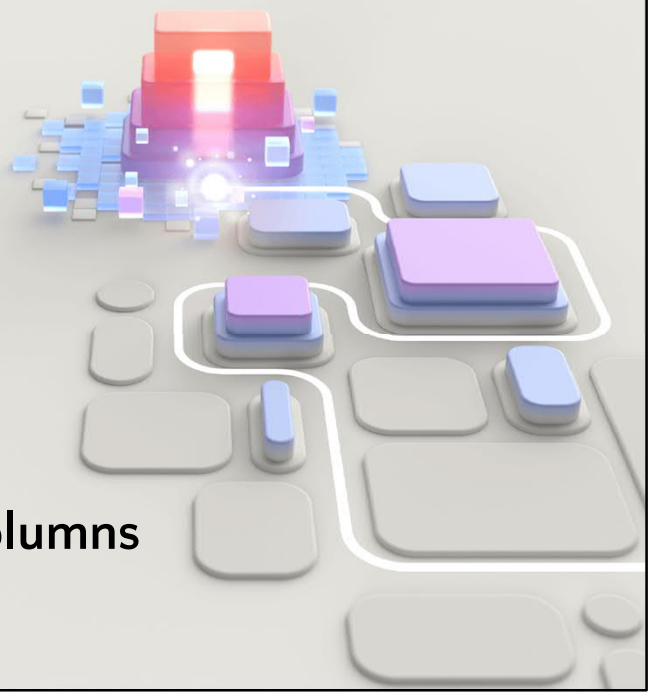
```
SELECT
    first_name,
    last_name,
    phone
FROM
    sales.customers
WHERE phone is null;
```



Why???

https://www.sqlservertutorial.net/sql-server-indexes/sql-server-filtered-indexes/

Microsoft



Index with Computed Columns

Index with Computed Columns

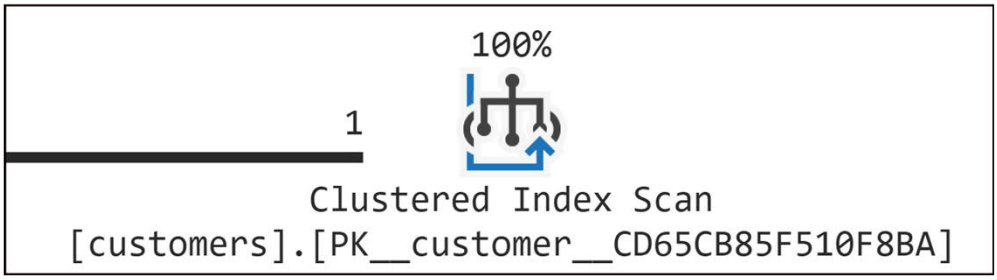
- Improve query performance when queries filter, sort, or join based on **computed expressions**.
- Allows **deterministic and precise computed columns** to be indexed like regular columns.
- Useful when frequently querying **calculated values** derived from table columns, avoiding repeated computation.

Problem Domain:

- Queries depend on expressions like
  - *TotalPrice = Quantity * UnitPrice*
  - or string/date manipulations.
- Without an index, SQL Server must **compute the value on the fly** for every row resulting in slower queries.

Example: Query with computed column index

```
SELECT  first_name, last_name, email
FROM sales.customers
WHERE
    substring(email, 0, CHARINDEX('@', email, 0)) =
    garry.espinoza';
```



https://www.sqlservertutorial.net/sql-server-indexes/sql-server-indexes-on-computed-columns/v

Example (cont...)

```
ALTER TABLE sales.customers
   ADD email_alias AS
       SUBSTRING(email, 0, CHARINDEX('@',
email, 0) );
```

```
CREATE INDEX ix_email_alias
ON sales.customers(email_alias);
```

https://www.sqlservertutorial.net/sql-server-indexes/sql-server-indexes-on-computed-columns/

Example (cont…)

```
CREATE INDEX ix_email_alias
ON sales.customers(email_alias);
```

Updated Query using computed column:

```
SELECT  first_name, last_name, email
FROM    sales.customers
WHERE   email_alias = 'garry.espinoza';
```

https://www.sqlservertutorial.net/sql-server-indexes/sql-server-indexes-on-computed-columns/

Example (cont...)