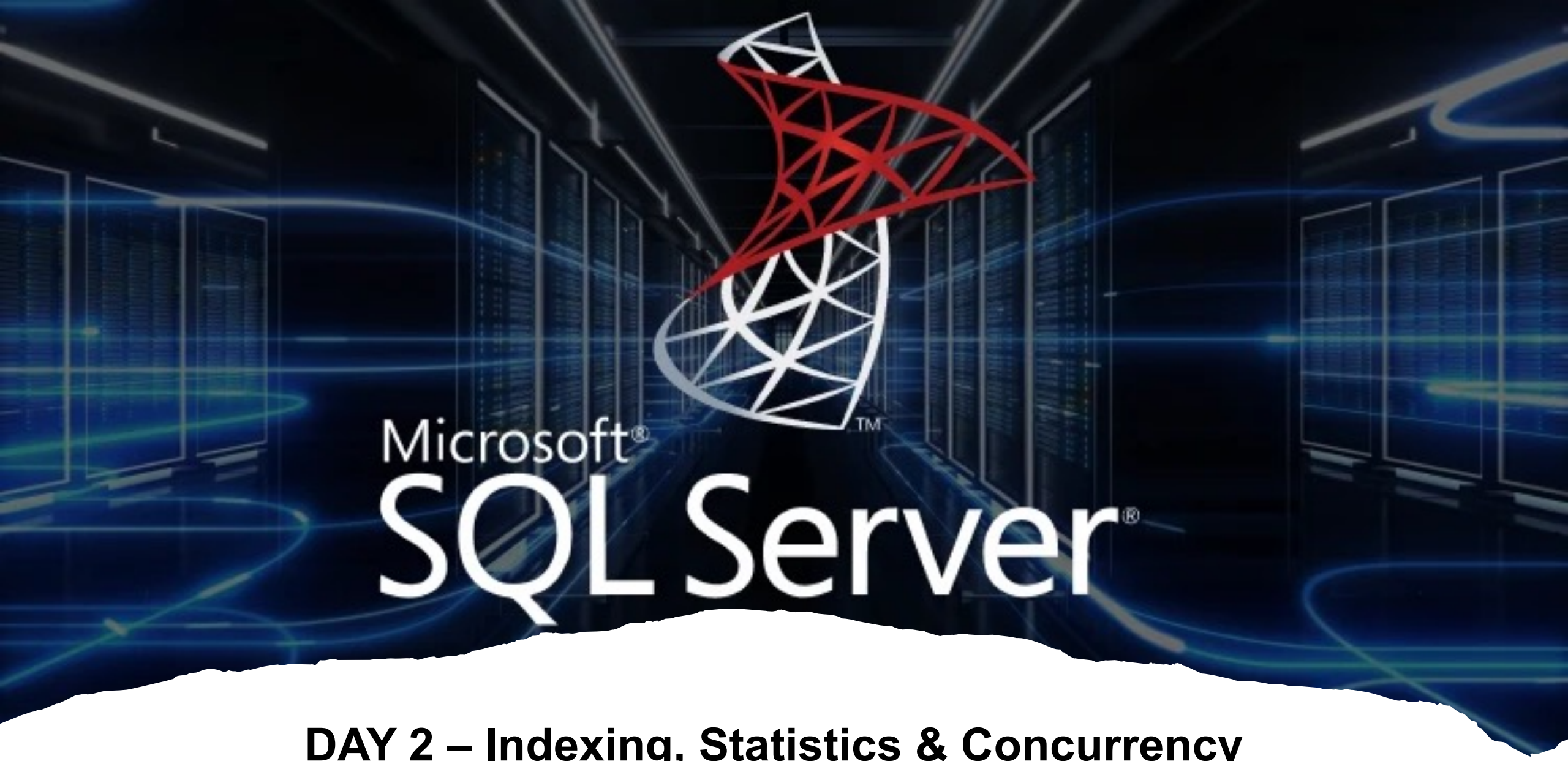


DAY 2 – Indexing, Statistics & Concurrency

Surendra Panpaliya

DAY 2

Module 3: Indexing, Statistics & Concurrency

Module 4: Wait Stats, Blocking & Concurrency

Module 3: Index & Statistics Tuning

OBJECTIVE

By the end of this module, you will:

Understand **what indexes really are**

Know **why some queries are fast and others are slow**

Learn how **statistics guide SQL Server decisions**

Stop blindly rebuilding indexes

Confidently decide **what to index, when, and why**

PART 1: WHAT IS AN INDEX?

Why Indexes Exist at All

Without Index

SELECT * FROM Employees WHERE emp_id = 101;

SQL Server must:

Read **every row**

Check condition one by one

Table Scan (slow for big tables)

With Index

SQL Server:

Jumps directly to the row

Index Seek (fast)

Index = Data structure that speeds up data access

Why DBAs Care About Indexes?

Indexes affect:

Query speed

CPU usage

Memory grants

Blocking

Why DBAs Care About Indexes?

Indexes affect:

Deadlocks

Storage

Indexes = performance + concurrency

Real-Life Example 📖

Imagine a **big textbook** 📚

- Without index → you scan every page
- With index → you jump directly to the topic

Indexes help SQL Server find data *FASTER*

How SQL Server Indexes Work ?

All indexes use a **B-Tree structure**:

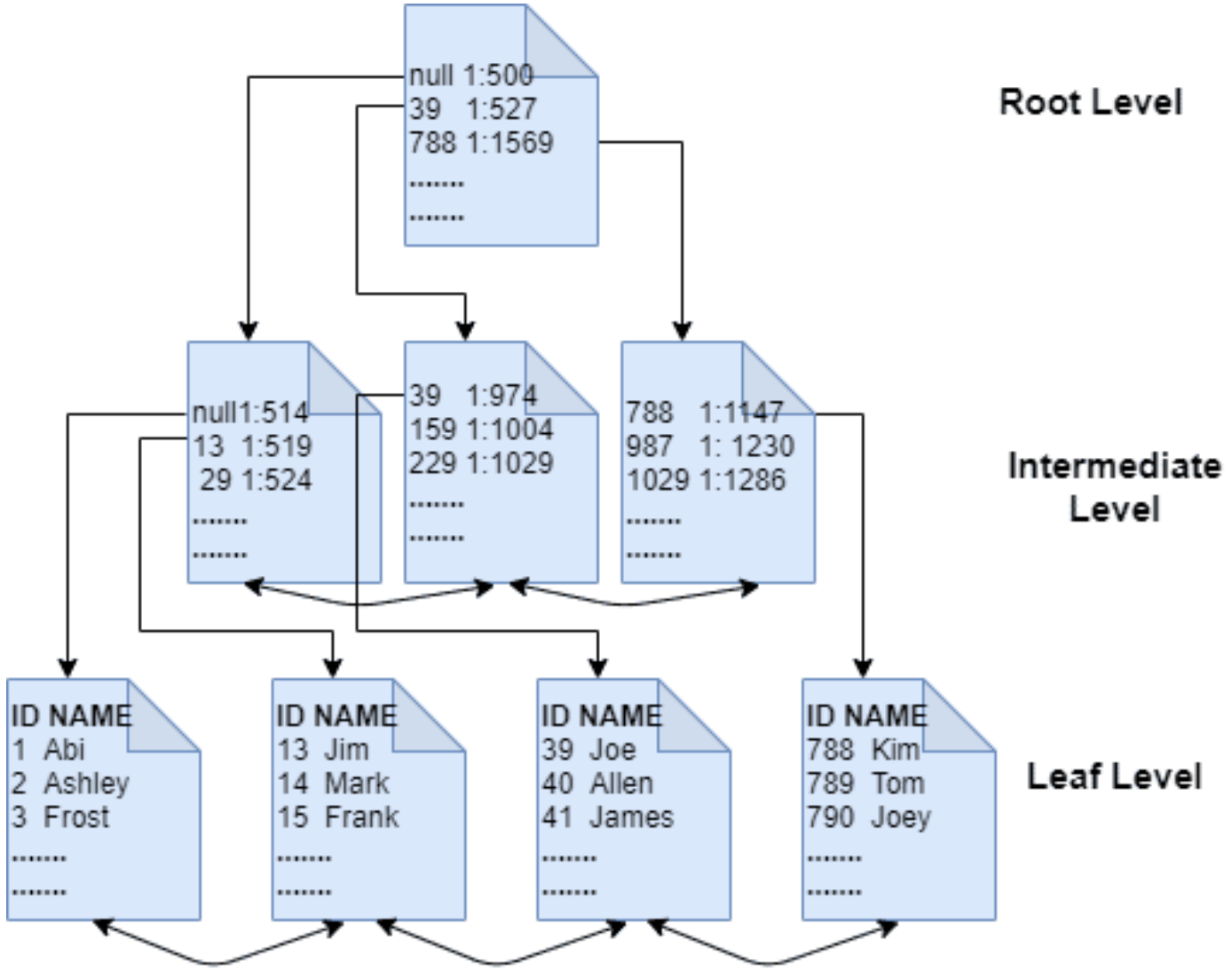
Root Page (entry point)

↓

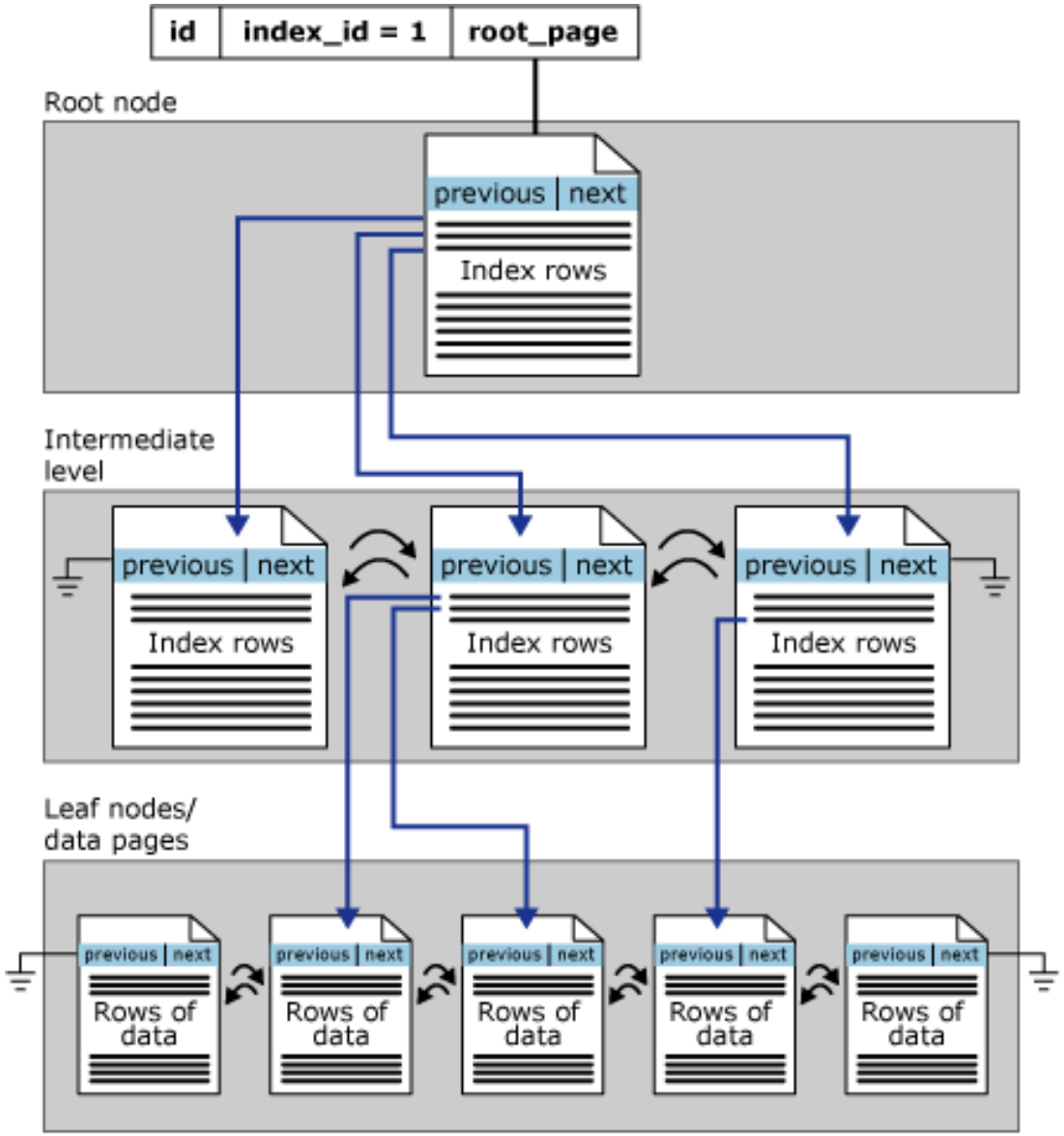
Intermediate Pages

↓

Leaf Pages (actual data or pointer to data)



B-Tree Clustered Index on ID


id
index_id = 1
root_page
Root node
previous | next
Index rows
Intermediate level
previous | next
Index rows
previous | next
Index rows
previous | next
Index rows
Leaf nodes/ data pages
previous | next
Rows of data
previous | next
Rows of data
previous | next
Rows of data
previous | next
Rows of data
previous | next
Rows of data

1. Clustered vs Non-Clustered Indexes

1.1 Clustered Index

What is it?

A **clustered index** decides **how data is physically stored** in the table.

Table data is stored **in the order of the clustered index key**

How data is stored ?

Table WITH clustered index = Data is stored

in sorted order of clustered key

CREATE CLUSTERED INDEX CI_Employees_ID ON Employees(emp_id);

How data is stored ?

Data on disk:

emp_id = 1

emp_id = 2

emp_id = 3

Key Rule

Only ONE clustered index per table

Why only ONE clustered index?

DATA ROWS THEMSELVES ARE THE **LEAF LEVEL**

A TABLE CAN HAVE ONLY ONE PHYSICAL ORDER

When to use Clustered Index ?

Choose column that is:

Unique (or near unique)

Narrow (INT preferred)

Ever-increasing (IDENTITY, date)

Frequently used in JOINs

When to use Clustered Index ?

Common choices:

entity_id (IDENTITY)

order_id

created_date

❌ Bad clustered index choices

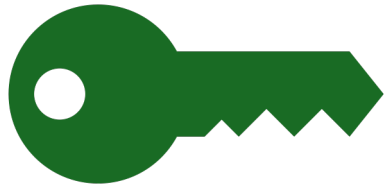
NVARCHAR columns

Columns that change often

Random GUIDs (without SEQUENTIALID)

DBA Insight

Clustered index affects **all nonclustered indexes**



Nonclustered indexes store clustered key as row locator

1.1 Clustered Index

Real-Life Example 🏠

Street with houses numbered 1,2,3,4

Houses are physically arranged in order

👉 Only **ONE clustered index per table**

Lab 1: Clustered Index

```
CREATE CLUSTERED INDEX IX_SalesOrderID
ON Sales.SalesOrderHeader (SalesOrderID);
```

Learning

- Data pages are reordered
- Faster range queries on that column

1.2 Non-Clustered Index

What is it?

A **non-clustered index** is a **separate structure** that points to the data.

What is a Nonclustered Index?

A separate structure that stores

index keys + pointers to the data.

You can have:

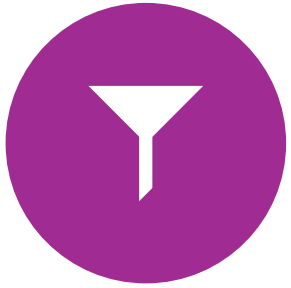
Multiple nonclustered indexes per table

Structure

Nonclustered Index

Leaf Level → Index key + Row Locator

Row locator = Clustered key (if clustered table)



RID (if heap)

Example

```
CREATE INDEX IX_Employees_Dept
```

```
ON Employees(department);
```

SQL Server

Uses index to find matching rows

Uses clustered key

to fetch full row (Key Lookup)

⚠️ Key Lookup

Occurs when:

Index does not contain all required columns

⚠️ Key Lookup

```
CREATE INDEX IX_Employees_Dept
ON Employees(department)
INCLUDE (emp_name, salary);
```

This is called a **covering index**

When to use Nonclustered Index

WHERE filters

JOIN columns

ORDER BY

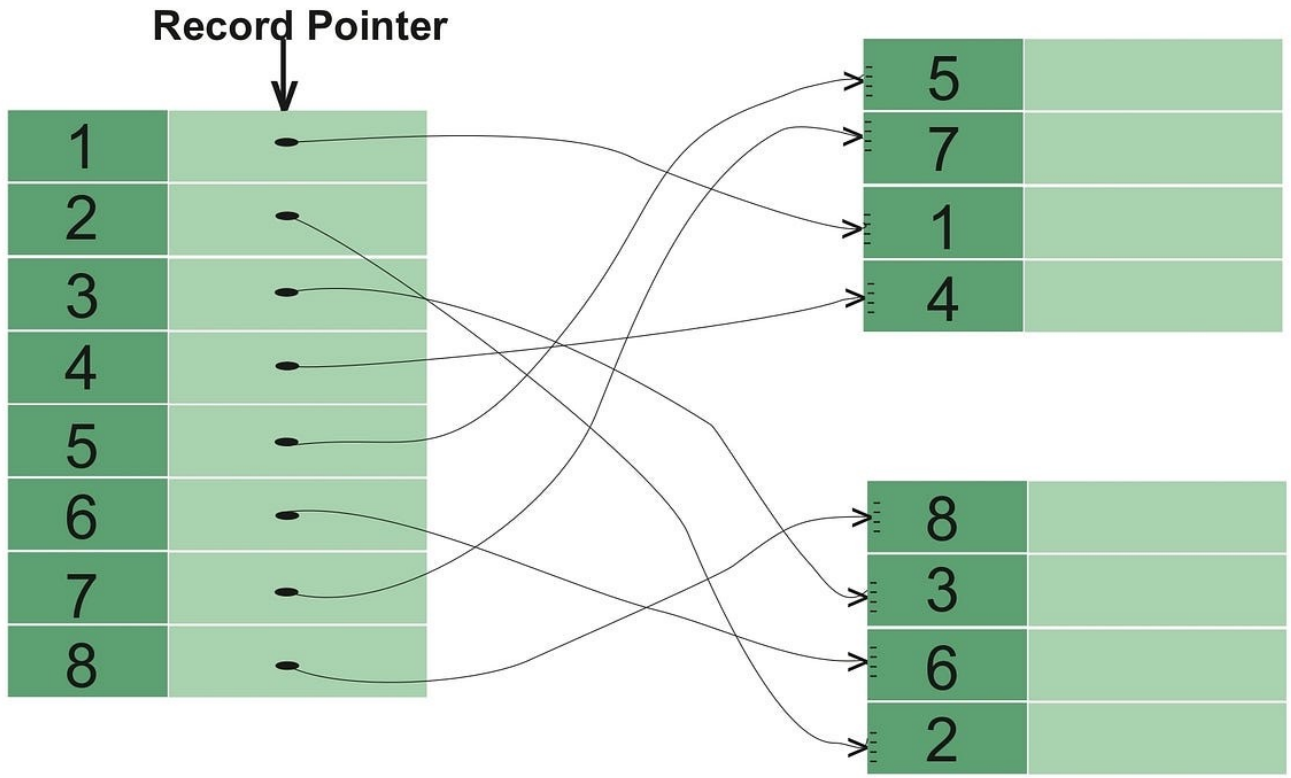
GROUP BY

1.2 Non-Clustered Index

Book index at the back

Topic → page number

You can have **many non-clustered indexes**



EXAMPLE OF NON-CLUSTERED INDEX

Lab 2: Non-Clustered Index

```
CREATE NONCLUSTERED INDEX IX_OrderDate
ON Sales.SalesOrderHeader (OrderDate);
```

Learning

- Helps queries filtering on OrderDate
- Does NOT change table storage

Summary

A clustered index defines

the physical order of data,

while nonclustered indexes are

separate structures used

to speed up queries.

Summary

Choosing the right index depends on

workload,

query patterns, and

data distribution.

Clustered vs Nonclustered Index

Feature	Clustered Index	Nonclustered Index
Definition	Defines physical order of table data	Separate structure with pointers to data
Number per table	Only 1	Multiple allowed
Leaf level	Actual data rows	Index key + row locator
Storage order	Sorted on disk	Logical order only
Speed	Fast for range queries	Fast for selective lookups
Key size impact	Very important (affects all NC indexes)	Less critical
Common use	Primary Key, IDENTITY	WHERE, JOIN, ORDER BY
Lookup behavior	No lookup needed	May need Key Lookup
Example	entity_id	country_code

Heap (Table Without Clustered Index)

A table with NO clustered index

Data:

Stored randomly

No order

Heap (Table Without Clustered Index)

Problems with heaps:

Slower range queries

Forwarded records

Harder maintenance

DBAs usually avoid heaps in OLTP

Heap vs Clustered Table

Feature	Heap (No CI)	Clustered Index Table
Data order	Random	Sorted
Range queries	Slow	Fast
Forwarded records	Possible	Not possible
Maintenance	Harder	Easier
DBA preference	❌ Avoid in OLTP	✅ Preferred

PART 2: COVERING & FILTERED INDEXES

2.1 Covering Index

An index
that contains **all columns needed by a query**

SQL Server does **not need to go back to table**

2.1 Covering Index

```
CREATE INDEX IX_Emp_Dept ON Employees(department)
INCLUDE (emp_name, salary);
```

2.1 Covering Index

```
SELECT emp_name, salary FROM Employees
WHERE department = 'IT';
```

Benefit

No key lookup

Less IO

Faster queries

Real Life Example

You carry:

Notebook

Pen

Calculator

No need to go back home.

Lab 3: Covering Index

```
CREATE NONCLUSTERED INDEX IX_Covering
ON Sales.SalesOrderDetail (ProductID)
INCLUDE (OrderQty, UnitPrice);
```

Covering Index vs Regular Nonclustered Index

Feature	Covering Index	Regular NC Index
INCLUDE columns	Yes	No
Key Lookup	❌ Avoided	⚠️ Possible
Performance	Faster	Slower if lookup
Storage	Slightly more	Less
DBA use	Reporting queries	Simple filters

2.2 Filtered Index

Index created on **subset of data**

Real-Life Example

Parking only for **electric vehicles**

2.2 Filtered Index

```
CREATE INDEX IX_Overdue
ON ComplianceFilings(due_date)
WHERE filing_status = 'OVERDUE';
```

Benefits:

SMALLER INDEX

FASTER SEEKS

LESS MAINTENANCE

Use when

Column has skewed data

Queries always filter on same value

Lab 4: Filtered Index

```
CREATE NONCLUSTERED INDEX IX_Filtered
ON Sales.SalesOrderDetail (OrderQty)
WHERE OrderQty = 1;
```

Filtered Index vs Full Index

Feature	Filtered Index	Full Index
Indexed rows	Subset only	All rows
Size	Smaller	Larger
Maintenance cost	Low	Higher
Use case	Skewed data (OVERDUE)	General use
Query requirement	WHERE must match filter	No restriction

Composite (Multi-column) Index

Index with multiple columns

CREATE INDEX IX_CF_Status_Due

ON ComplianceFilings(filing_status, due_date);

Composite (Multi-column) Index

Rule: Order matters

Equality first, range later:

```
WHERE status = 'OVERDUE'
```

```
AND due_date < GETDATE()
```

Composite Index vs Single Column Index

Feature	Composite Index	Single Column Index
Columns	Multiple	One
Column order matters	✅ Yes	❌ No
Supports complex filters	Yes	Limited
Example	(status, due_date)	(status)

PART 3: COLUMNSTORE INDEXES (WHEN TO USE)

Columnstore Index (Analytics)

Stores data

column-wise,

not row-wise

Columnstore Index (Analytics)

Best for:

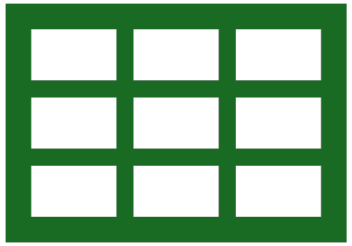
- Reporting
- Aggregations
- Data Warehouses

Types

Clustered Columnstore

Nonclustered Columnstore

Not good for



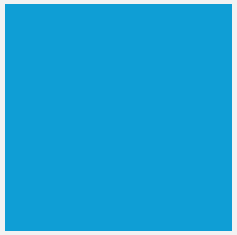
❌ Heavy OLTP updates

❌ Single-row lookups

Real-Life Example

Excel:

All values of one column together



Great for reports and analytics

When to Use Columnstore

✓ Large tables

✓ Reporting / analytics

✓ Aggregations (SUM, COUNT)

✗ Small OLTP tables

✗ Frequent single-row updates

Lab 5: Columnstore Index

```
CREATE CLUSTERED COLUMNSTORE INDEX CCI_Sales
ON Sales.SalesOrderDetail;
```

Columnstore vs Rowstore Index

Feature	Columnstore Index	Rowstore Index
Storage	Column-wise	Row-wise
Best for	Analytics, reporting	OLTP
Aggregations	Very fast	Slower
Updates	Expensive	Fast
Compression	Very high	Moderate
Example workload	Data warehouse	Transactions

Unique Index

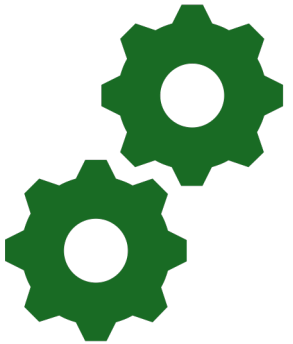
Enforces uniqueness

CREATE UNIQUE INDEX UX_Email

ON Users(email);

Benefit

Data integrity



Helps optimizer

Unique Index vs Non-Unique Index

Feature	Unique Index	Non-Unique Index
Duplicate values	❌ Not allowed	Allowed
Data integrity	Enforced	Not enforced
Optimizer benefit	High	Normal
Example	Email, PAN	Country code

XML / Spatial / Full-Text Index (Specialized)

Type	Used for
XML Index	XML columns
Spatial	Geography / geometry
Full-Text	Text search

When to Use Which Index

Scenario	Recommended Index
Primary Key	Clustered Index
Frequent WHERE filter	Nonclustered Index
Reporting query	Covering Index
Skewed status values	Filtered Index
Large aggregations	Columnstore Index
Enforce uniqueness	Unique Index
Multi-condition filters	Composite Index

PART 4: INDEX FRAGMENTATION

Common Myth ❌

Fragmentation = slow performance always

Reality

Fragmentation affects **range scans**

Does NOT always affect seeks

PART 4: INDEX FRAGMENTATION

Real-Life Example 

Slight potholes on highway

Car still runs fine

PART 4: INDEX FRAGMENTATION

Rebuild vs Reorganise (Simple Rule)

Fragmentation	Action
< 5%	Do nothing
5–30%	Reorganize
> 30%	Rebuild

Lab 6: Check Fragmentation

```
SELECT index_id, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats
(DB_ID(), OBJECT_ID('Sales.SalesOrderDetail'), NULL, NULL,
'SAMPLED');
```

Index Selection Decision Flowchart

START

|

|-- Is this a table with NO clustered index?

| |

| |-- YES → Create CLUSTERED INDEX

| | (PK / IDENTITY / date)

| |

| |-- NO

Index Selection Decision Flowchart

START

|

|-- Is query slow?

| |

| |-- NO → STOP (no index needed)

| |

| |-- YES

|

|-- Does query filter on WHERE clause?

Index Selection Decision Flowchart

START

|

|-- Does query filter on WHERE clause?

| |

| |-- YES → NONCLUSTERED INDEX on filter column

| |

| |-- NO

|

|-- Does query JOIN on a column?

Index Selection Decision Flowchart

START

|

|-- Does query JOIN on a column?

| |

| |-- YES → NONCLUSTERED INDEX on JOIN column

| |

| |-- NO

|

|-- Does query use ORDER BY / GROUP BY?

Index Selection Decision Flowchart

START

|

|-- Does query use ORDER BY / GROUP BY?

| |

| |-- YES → Index matching ORDER BY / GROUP BY

| |

| |-- NO

|

|-- Do you see KEY LOOKUP?

Index Selection Decision Flowchart

START

|

|-- Do you see KEY LOOKUP?

| |

| |-- YES → Add INCLUDE columns (Covering Index)

|

|-- Is filter always on same value?

Index Selection Decision Flowchart

START

|

|-- Is filter always on same value?

| |

| |-- YES → FILTERED INDEX

|

|-- Is query analytical (COUNT, SUM, millions rows)?

| |

| |-- YES → COLUMNSTORE INDEX

END

Quick DBA Cheat Sheet

Scenario	Index Choice
Primary key	Clustered
WHERE filter	Nonclustered
JOIN column	Nonclustered
Key Lookup	Covering
Fixed filter	Filtered
Reporting	Columnstore
Heap table	Add clustered

PART 5: FILL FACTOR STRATEGY

What is Fill Factor?

Leave some empty space
in index pages

Real-Life Example

Don’t pack suitcase 100%
→ space for souvenirs

What is Fill Factor?

When to Use Lower Fill Factor

Frequent inserts

Frequent updates

PART 5: FILL FACTOR STRATEGY

Simple Example

CREATE INDEX IX_FillFactor

ON Sales.SalesOrderDetail (ProductID)

WITH (FILLFACTOR = 80);

PART 6: STATISTICS

Tell SQL Server:

How many rows exist

How data is distributed

Optimizer uses stats to choose **best plan**

6.1 Histograms

A histogram shows **value distribution** in a column.

Real-Life Example

Population distribution by age group

Lab 7: View Histogram

DBCC SHOW_STATISTICS ('Sales.SalesOrderDetail', 'IX_ProductID’);

Observe:

- Range values
- Density
- Frequency

6.2 Auto-Update Thresholds

SQL Server updates statistics
when **enough data changes**

- Small tables → frequent updates
- Large tables → delayed updates

6.3 Multi-Column Statistics

Statistics on

multiple columns together

Used when queries filter

on **column combinations**

6.3 Multi-Column Statistics

```
CREATE STATISTICS Stat_Multi
ON Sales.SalesOrderDetail (ProductID, OrderQty);
```

PART 7: HANDS-ON LABS (PRACTICAL)

Lab 8: Find Missing Indexes

```
SELECT *  FROM sys.dm_db_missing_index_details;
```

Lab 9: Index Usage Analysis

```
SELECT * FROM sys.dm_db_index_usage_stats
WHERE object_id = OBJECT_ID('Sales.SalesOrderDetail');
```

Lab 10: Fix Query Using Statistics

- Run slow query
- Update statistics
- Rerun query
- Compare execution plan

```
UPDATE STATISTICS Sales.SalesOrderDetail;
```

MEMORY CHEAT SHEET

Concept	Simple Meaning
Clustered Index	Physical order
Non-Clustered Index	Lookup map
Covering Index	Everything in one place
Filtered Index	Partial data index
Columnstore	Analytics index

MEMORY CHEAT SHEET

Concept	Simple Meaning
Fragmentation	Page disorder
Fill Factor	Empty space
Statistics	Optimizer knowledge
Histogram	Data distribution

Surendra Panpaliya

Founder and CEO

GKTCS Innovations

https://www.gktcs.com