SQL Server Indexing – Beginner to Senior-Level Notes

For revision & interviews (copy-paste friendly)
Prepared for **Nazif**

---

1. How SQL Server Stores Data (Foundation)

- SQL Server **never reads rows**, it reads **pages**
- **Page size = 8 KB**
- A page contains multiple rows
- Disk read is expensive → goal is to read **fewer pages**

Performance rule:

> Faster query = fewer pages read

---

2. What Happens Without Index (Table Scan)

```
SELECT * FROM Products WHERE Id = 50000;
```

- SQL Server does not know where the row is
- Reads **every page** one by one
- This is called **TABLE SCAN**
- Slow for large tables

---

3. Why Index Exists

Index answers one question:

> "Which page contains my data?"

- Index is a **separate structure**
- Smaller than table
- Stored in **sorted order**
- Helps SQL Server jump directly to the page

---

4. Index Internal Structure (B-Tree)

Every index in SQL Server is a **B-Tree**

```
Root Page
   ↓
Intermediate Pages
   ↓
Leaf Pages
```

- Balanced tree
- Search needs only **2–4 page reads**

---

5. Clustered Index (Most Important)

Definition

> Clustered index = **actual table data stored in sorted order**

- Leaf pages = real data pages
- Only **ONE clustered index per table**

Example

```
CREATE CLUSTERED INDEX PK_Products
ON Products(Id);
```

When SQL Server Searches

```
WHERE Id = 250
```

- Uses B-Tree
- Jumps directly to correct page
- This is an **INDEX SEEK**

Best Columns for Clustered Index

- Primary key
- Identity / sequential column
- Stable & unique

❌Do NOT use clustered index on: - Name - Email - Status

---

6. Non-Clustered Index

Definition

> Non-clustered index stores **column value + pointer to row**

- Separate from table
- Can have **many non-clustered indexes**

Example

```
CREATE NONCLUSTERED INDEX IX_Products_Category
ON Products(CategoryId);
```

Pointer Type

Table Type	Pointer
Clustered table	Clustered key
Heap table	RID (File:Page:Slot)

---

7. Key Lookup (Performance Killer)

Query:

```
SELECT Name, Price
FROM Products
WHERE CategoryId = 5;
```

- Index used only for CategoryId
- Name & Price not in index
- SQL Server goes back to table
- This extra step = **KEY LOOKUP**

Problem: - Many rows → many random reads - Slow execution

---

8. Covering Index (Solution)

```
CREATE NONCLUSTERED INDEX IX_Products_Category
ON Products(CategoryId)
INCLUDE (Name, Price);
```

- Leaf page contains all needed columns

- No key lookup
- Faster queries

---

9. Heap Table (No Clustered Index)

- Table without clustered index
- Rows stored randomly
- Pointer = RID

Problems: - Fragmentation - Slow reads

Use only for: - Temp tables - Staging tables

---

10. Index Seek vs Index Scan

Index Seek (GOOD)

- Uses index order
- Reads few pages

Index Scan (BAD)

- Reads all pages
- Happens when:
- No index
- Wrong index
- Functions in WHERE clause

---

11. LIKE Search & Index

❌BAD (index not used)

```
WHERE Name LIKE '%phone%'
```

✅GOOD

```
WHERE Name LIKE 'phone%'
```

Index:

```
CREATE NONCLUSTERED INDEX IX_Products_Name
ON Products(Name);
```

12. Functions in WHERE (Avoid)

❌BAD

```
WHERE YEAR(OrderDate) = 2024
```

✅GOOD

```
WHERE OrderDate >= '2024-01-01'
AND OrderDate < '2025-01-01'
```

Reason: Functions break index order (non-sargable)

---

13. Page Split (Why Index Becomes Slow)

- Page size = 8 KB
- Page becomes full
- Insert happens in middle

SQL Server: 1. Creates new page 2. Moves ~50% rows 3. Updates pointers

Result: - Fragmentation - Extra IO

---

14. Fill Factor

```
FILLFACTOR = 80
```

- Page filled only 80%
- 20% free space

✔️Fewer page splits ❌More disk usage

---

15. Index Fragmentation

Type	Meaning
Logical	Page order broken
Physical	Pages far apart

Fix: - REORGANIZE (light) - REBUILD (heavy)

---

16. Statistics (Optimizer Brain)

- Statistics tell SQL Server:
- Row count
- Data distribution

Bad statistics → bad execution plan

Indexes without stats are useless

---

17. Too Many Indexes (Bad)

Each INSERT / UPDATE: - Updates clustered index - Updates all non-clustered indexes

Trade-off: ✔️Faster reads ❌Slower writes

---

18. What to Use: Clustered vs Non-Clustered

Golden Rule

- **Clustered index** → Primary key / data order
- **Non-clustered index** → Searching & filtering

Decision Table

Scenario	Index
Primary key	Clustered
Foreign key	Non-clustered
Search column	Non-clustered
Pagination	Clustered
Reporting	Covering non-clustered

---

19. Real-World Index Examples

User Login

```
CREATE UNIQUE NONCLUSTERED INDEX IX_Users_Email
ON Users(Email);
```

Product Search

```
CREATE NONCLUSTERED INDEX IX_Products_Search
ON Products(CategoryId, Price)
INCLUDE (Name, ImageUrl);
```

Order History

```
CREATE NONCLUSTERED INDEX IX_Orders_User_Date
ON Orders(UserId, OrderDate DESC);
```

---

20. Senior Developer One-Line Rules

- Clustered index defines physical order
- Non-clustered index stores pointers
- Seek is good, scan is bad
- Key lookup is expensive
- Covering index avoids lookup
- Index for queries, not tables

FINAL MEMORY LINE

> **SQL Server performance = number of pages read**

Indexes exist only to reduce page reads.

END OF NOTES