



DATA WITH BARAA

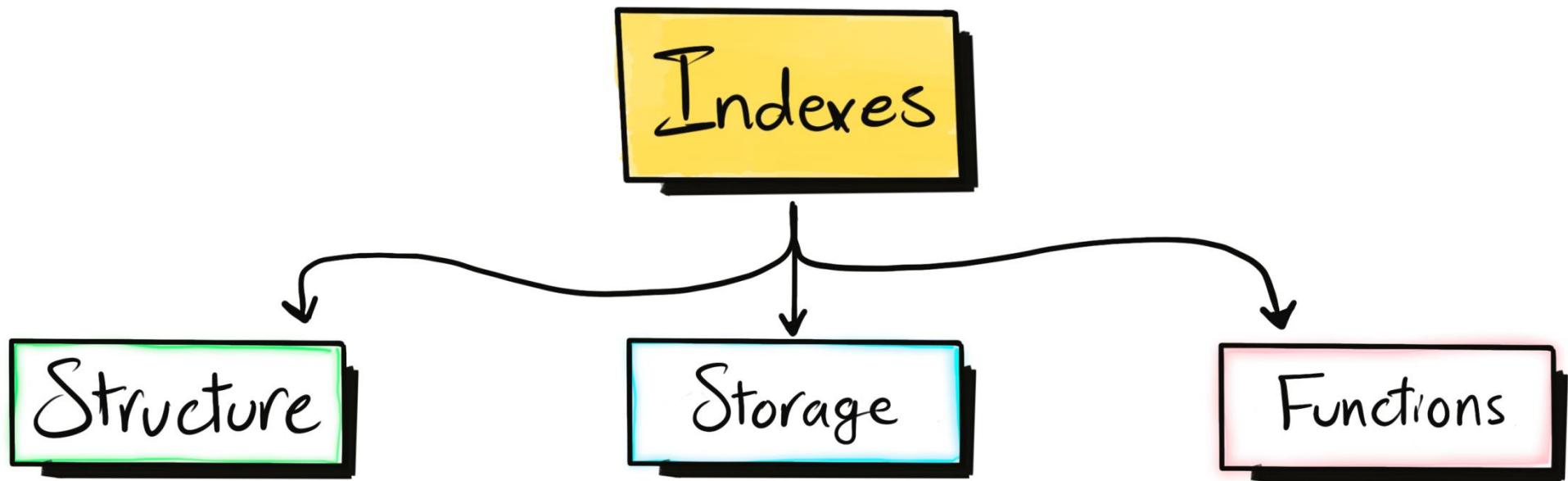
# Indexes

Baraa Khatib Salkini  
YouTube | **DATA WITH BARAA**  
SQL Course | Indexes



# INDEX

Data structure provides quick access to data,  
optimizing the speed of your queries.



Clustered Index

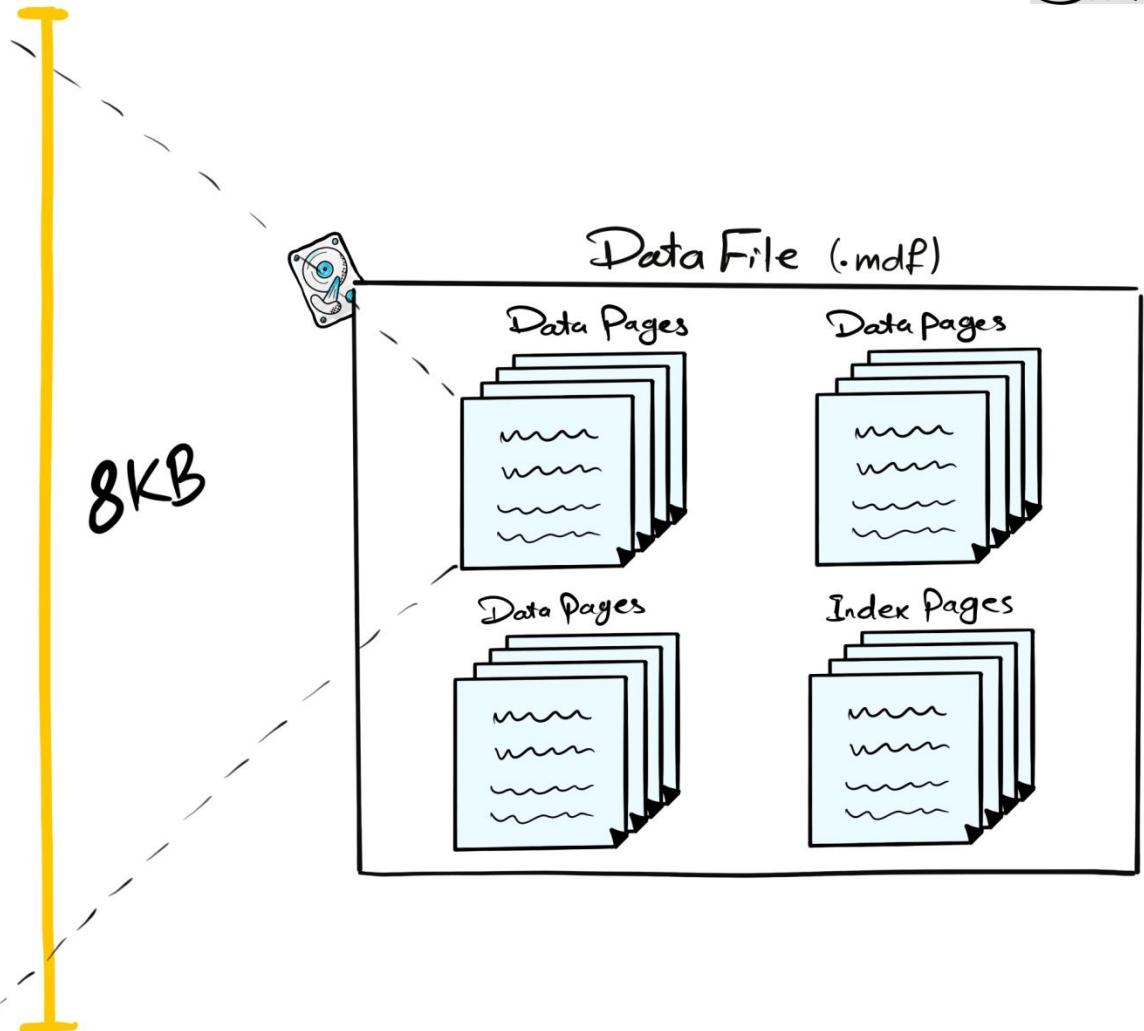
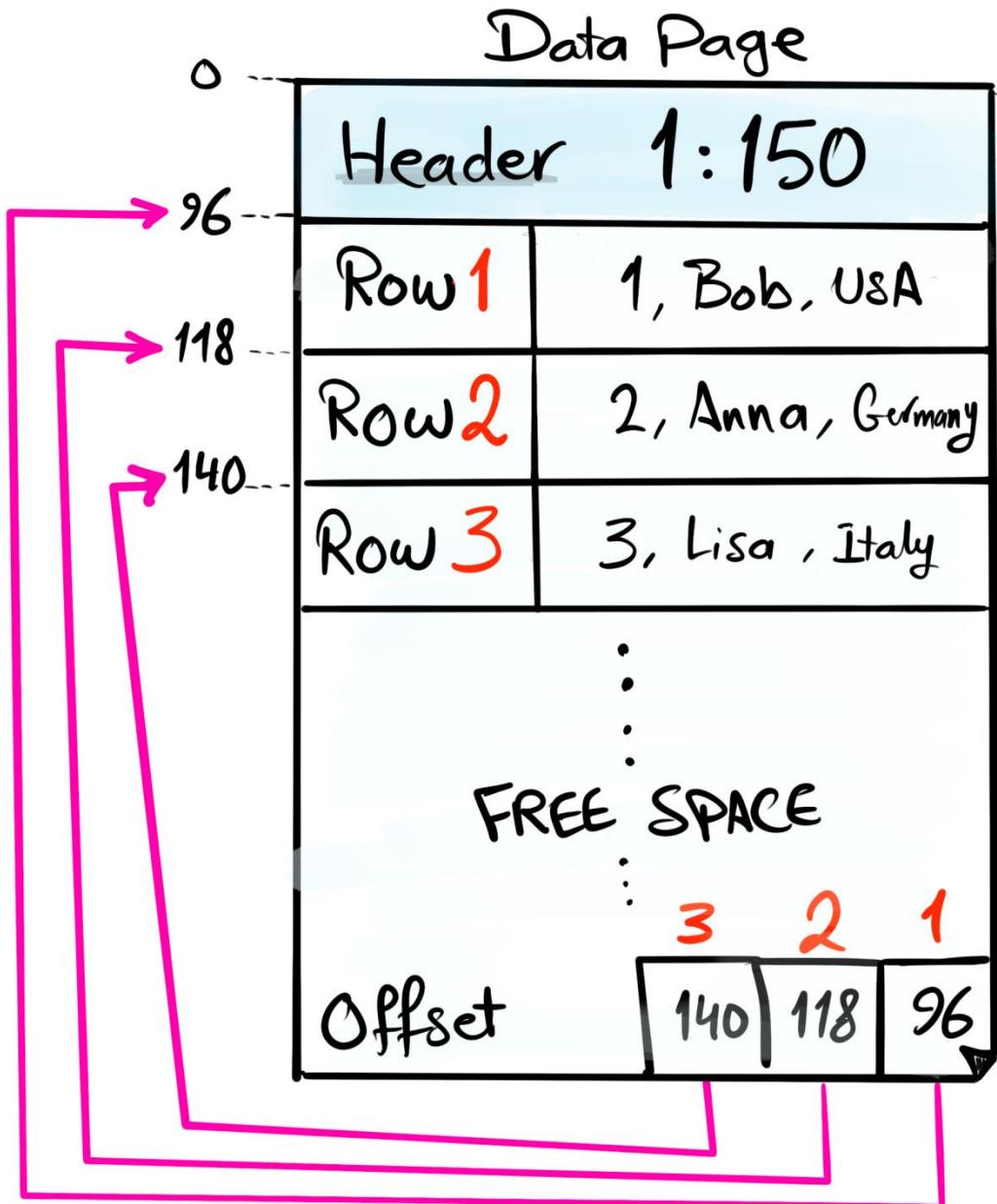
Non-Clustered Index

Rowstore Index

Columnstore Index

Unique Index

Filtered Index



# HEAP

1:100
12, Lisa
5, Chris
15, Mathew
6, Jessica
7, David

Data Page

1:101
20, Jane
2, Anna
3, Emily
9, Robert
19, Andrew

Data Page

1:102
1, Bob
4, John
8, Laura
10, Olivia
11, James

Data Page

1:103
18, Sarah
16, Emma
13, Daniel
14, Sophia
17, Brian

Data Page



# Data File (.mdf)

HEAP

①

1:100	
12	Lisa
5	Chris
15	Mathew
6	Jessica
7	David

Data Page

②

1:101	
20	Jane
2	Anna
3	Emily
9	Robert
19	Andrew

Data Page

③

1:102	
1	Bob
4	John
8	Laura
10	Olivia
11	James

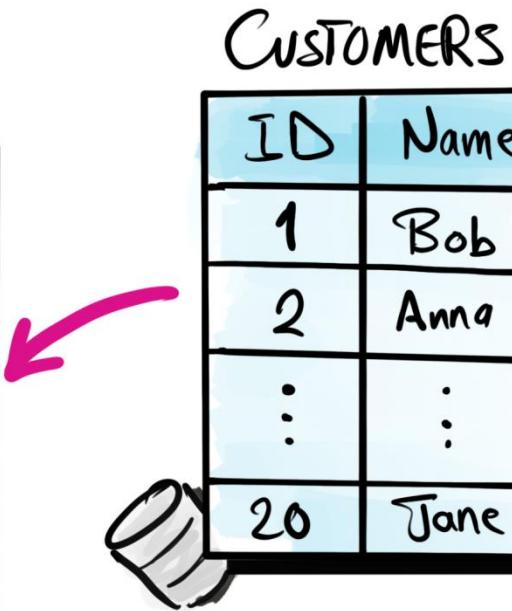
Data Page

④

1:103	
18	Sarah
16	Emma
13	Daniel
14	Sophia
17	Brain

Data Page

Full Table Scan



Sophia



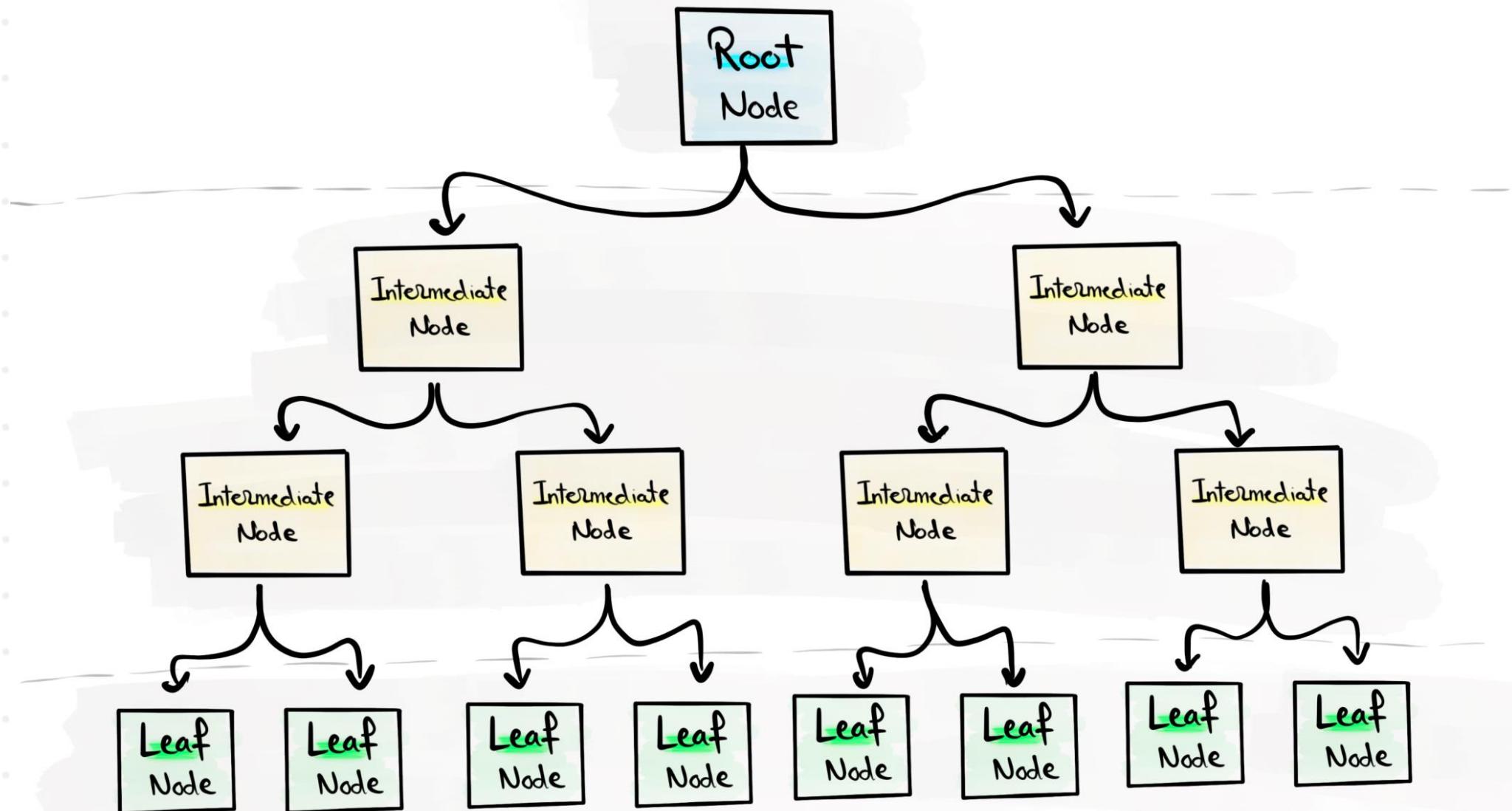
DATA WITH BARAA

# **B-TREE**

(BALANCE TREE)

Hierarchical structure storing data at leaves,  
to help quickly locate data.

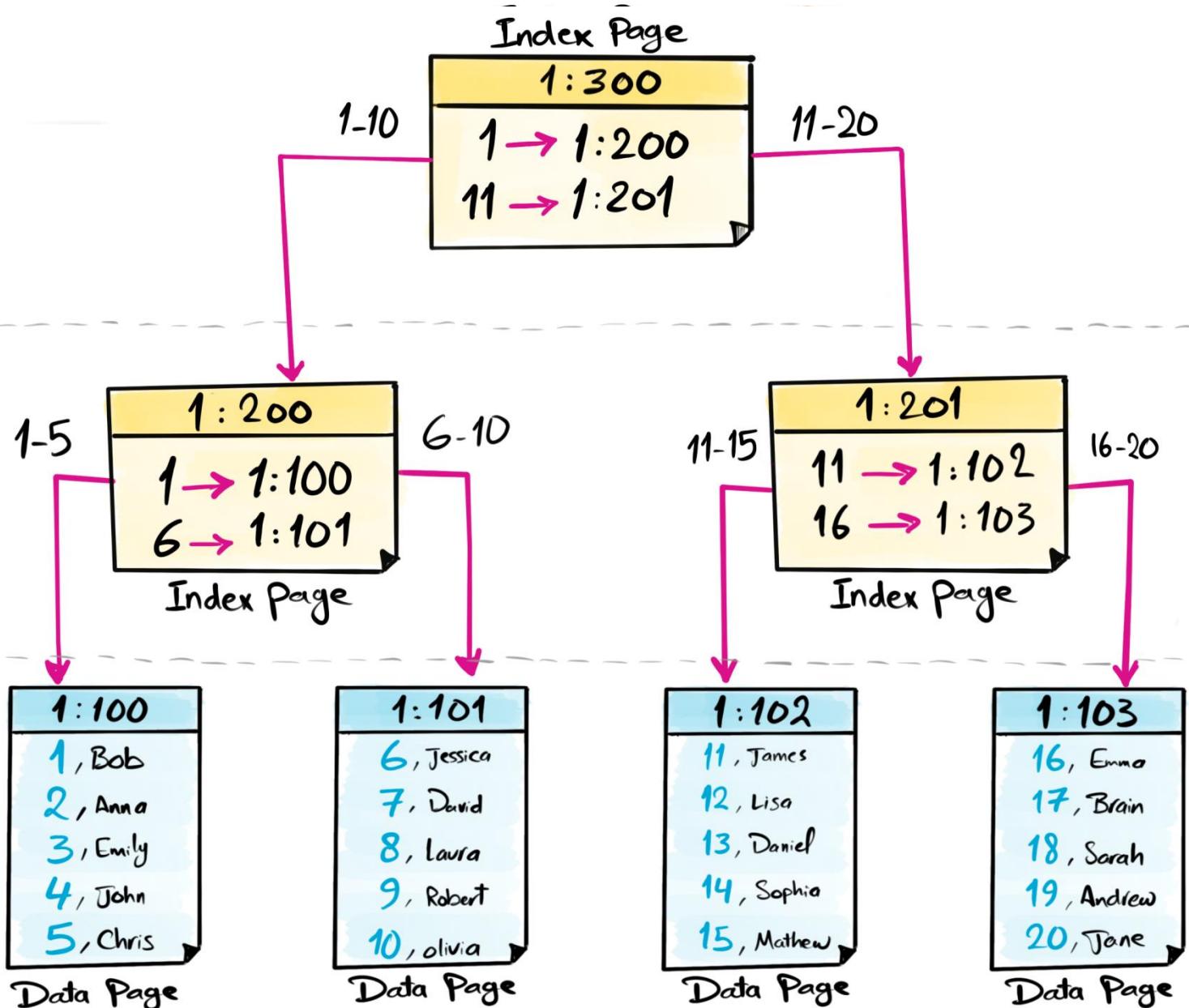
# B-TREE



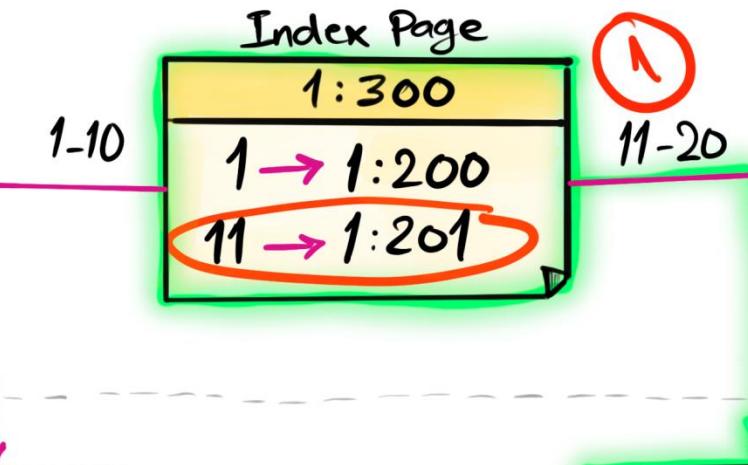
Root  
Node

Intermediate  
Nodes

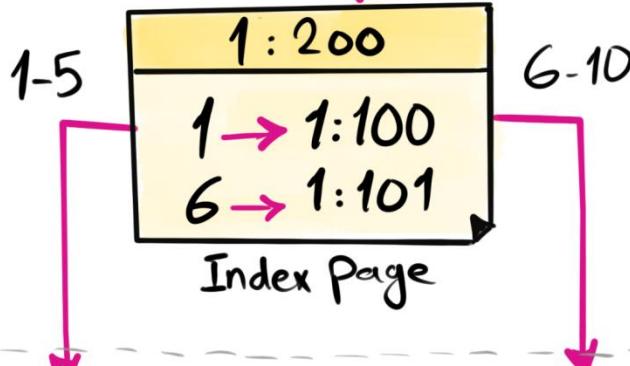
Leaf Level  
Nodes



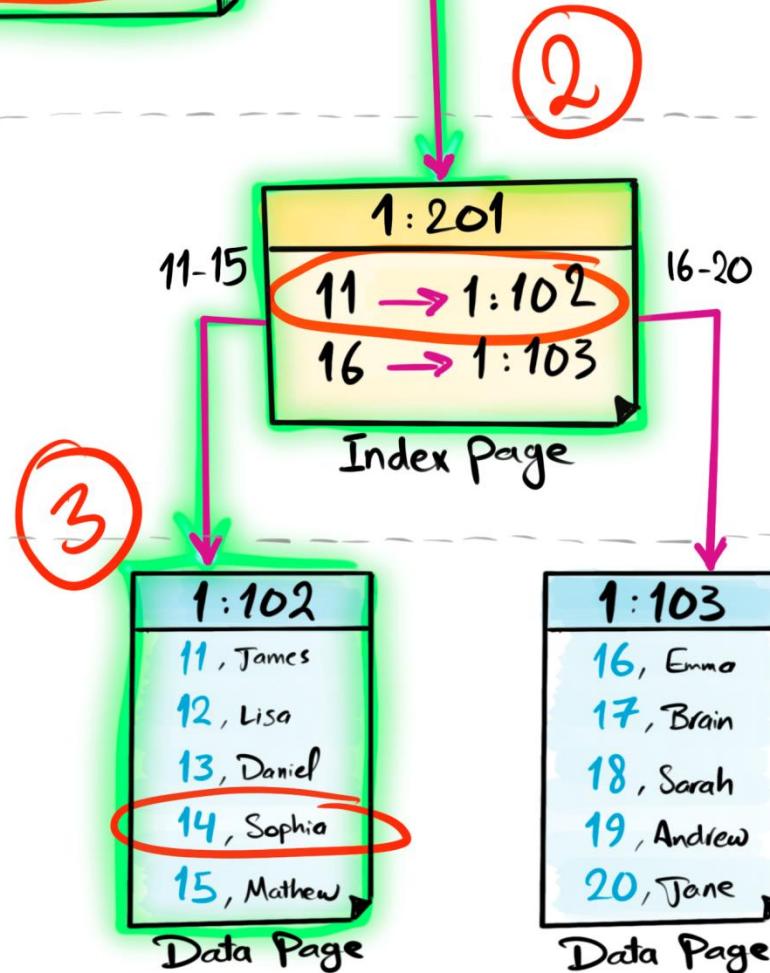
## Root Node



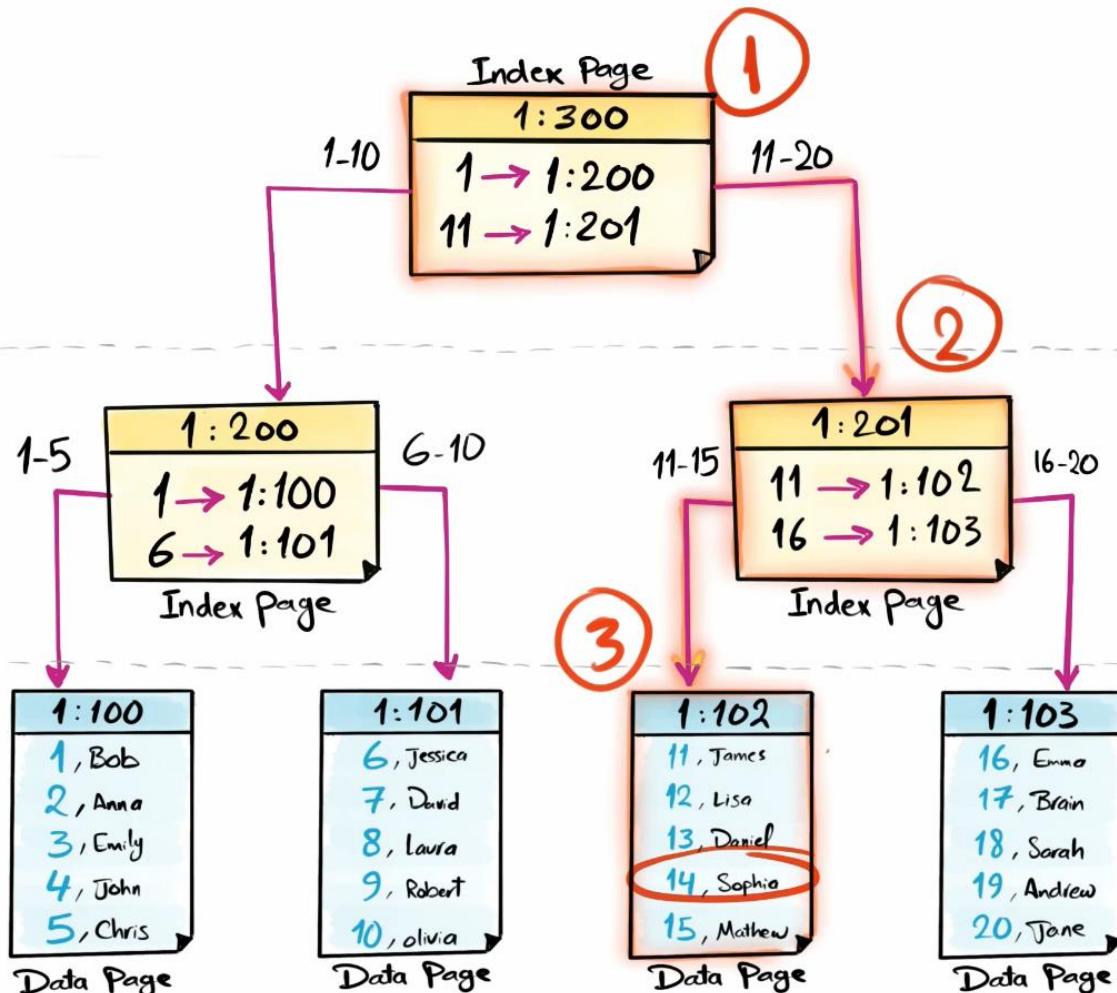
## Intermediate Nodes



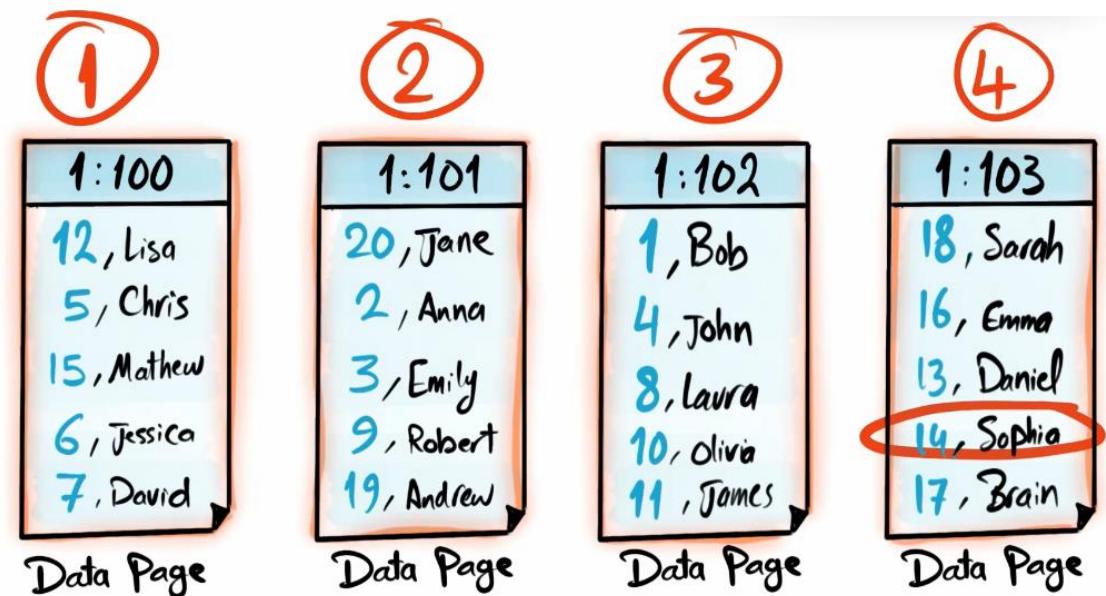
## Leaf Level

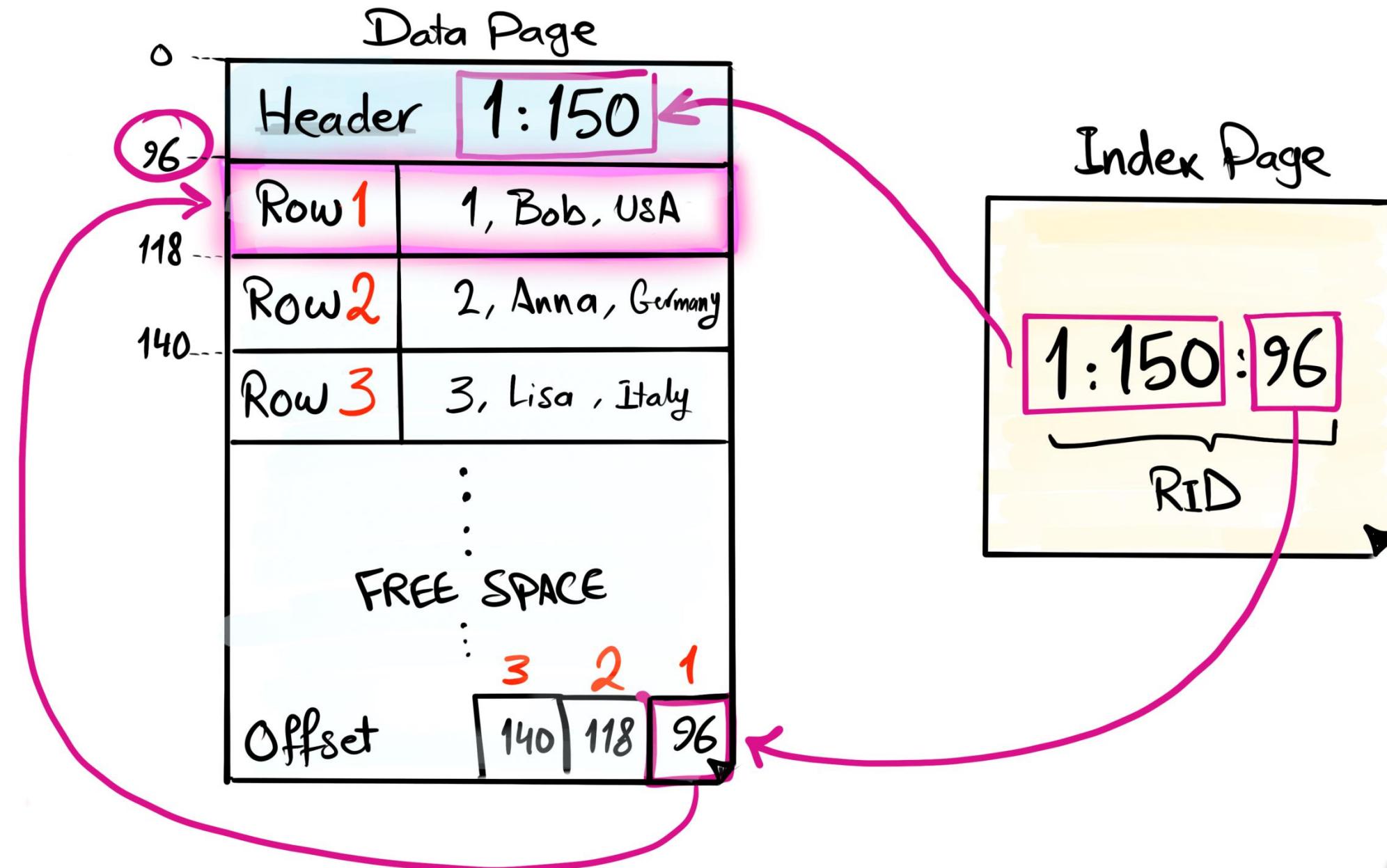


# Clustered INDEX



# Heap structure





## Index Page

1:200
1 → 1:102:96
2 → 1:101:140
3 → 1:101:188
4 → 1:102:140
5 → 1:100:140

## Index Page

1:201
6 → 1:100:250
7 → 1:100:380
8 → 1:102:188
9 → 1:101:250
10 → 1:102:250

## Index Page

1:202
11 → 1:102:380
12 → 1:100:96
13 → 1:103:188
14 → 1:103:250
15 → 1:100:188

## Index Page

1:203
16 → 1:103:188
17 → 1:103:380
18 → 1:103:96
19 → 1:101:380
20 → 1:101:96

## Data Page

1:100
12, Lisa
5, Chris
15, Mathew
6, Jessica
7, David

## Data Page

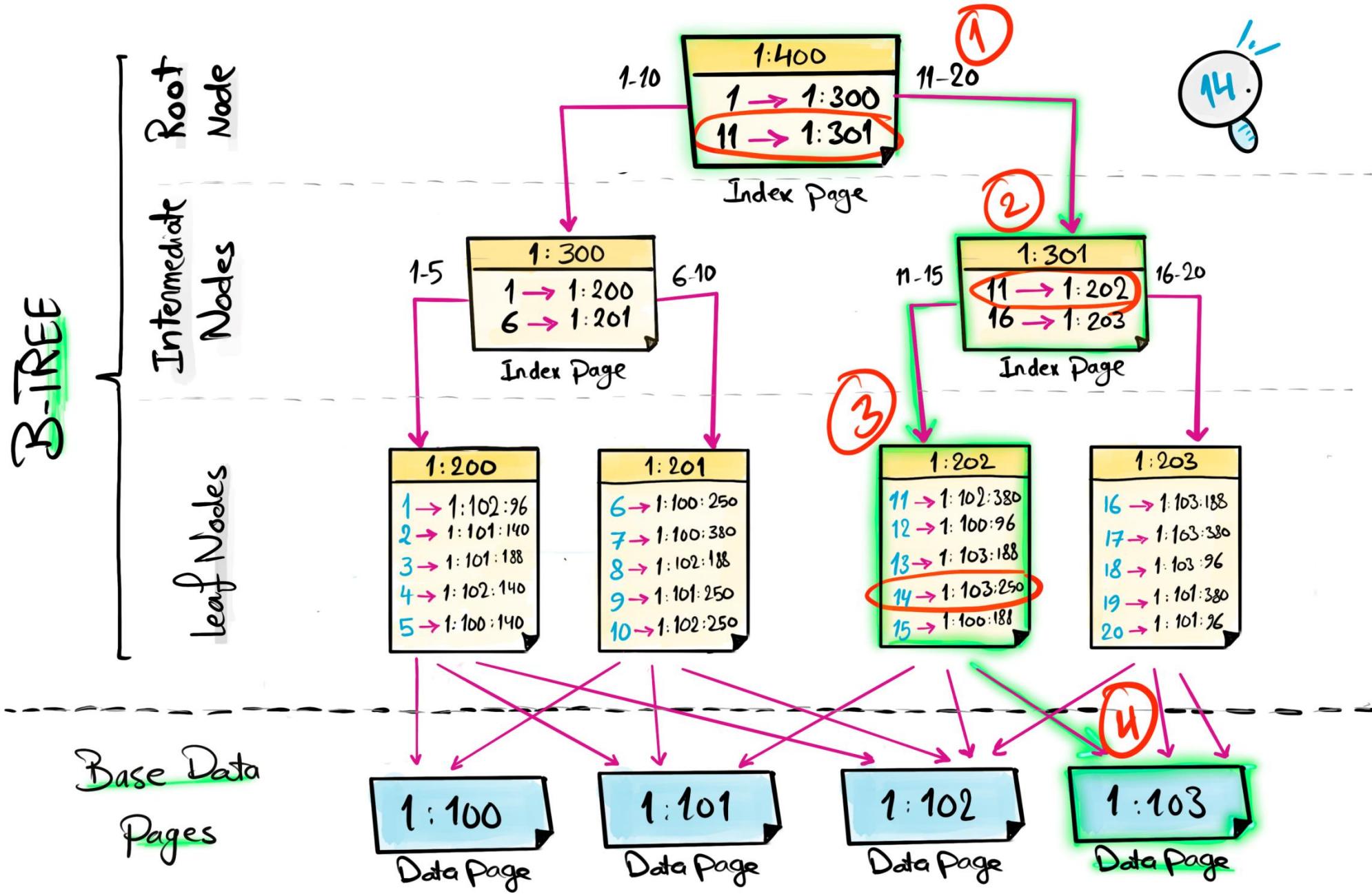
1:101
20, Jane
2, Anna
3, Emily
9, Robert
19, Andrew

## Data Page

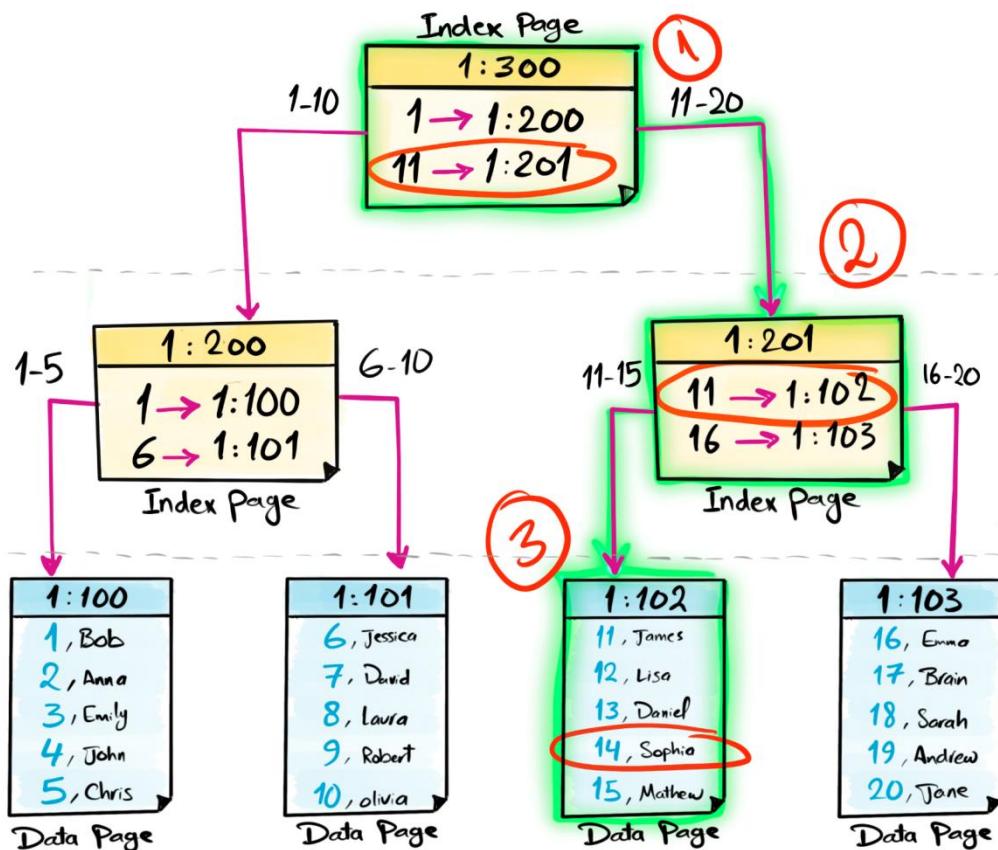
1:102
1, Bob
4, John
8, Laura
10, Olivia
11, James

## Data Page

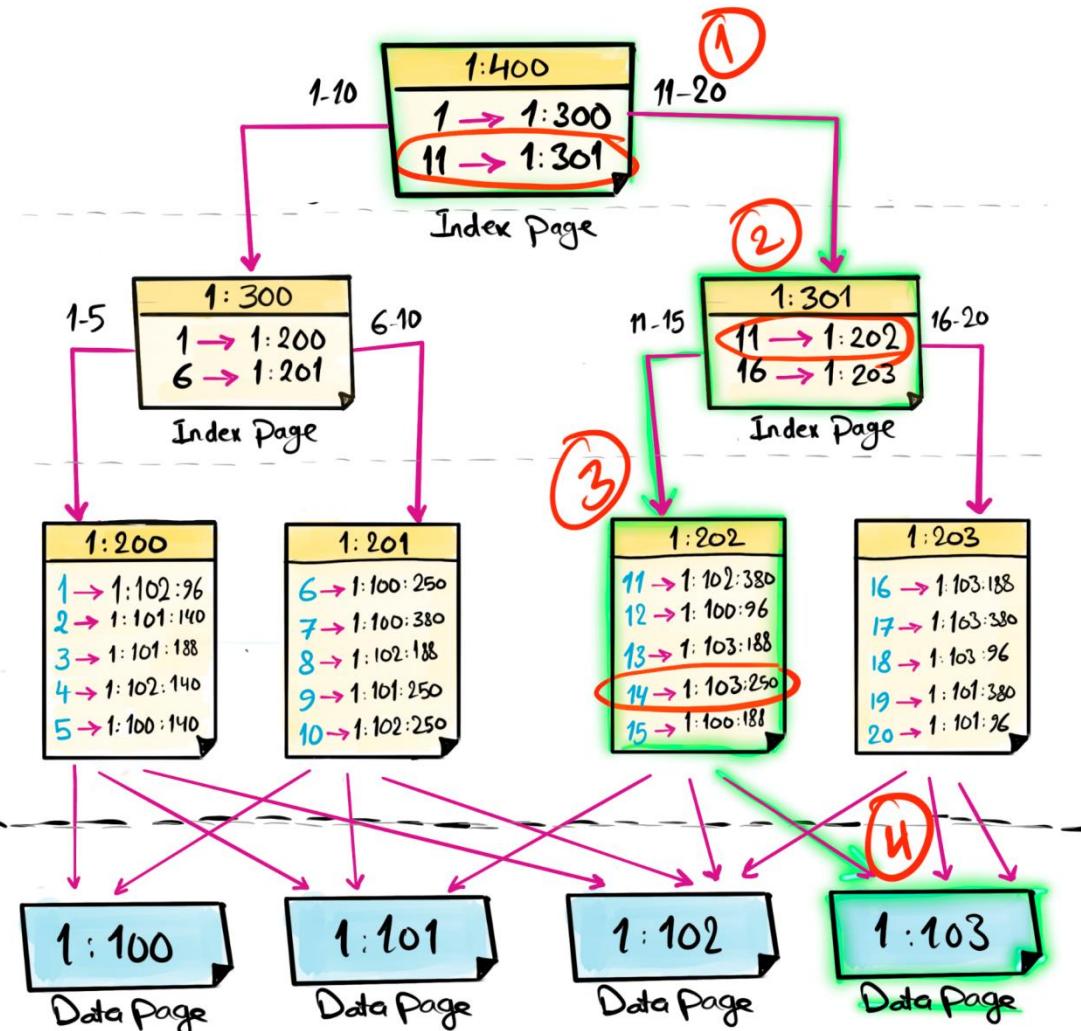
1:103
18, Sarah
16, Emma
13, Daniel
14, Sophia
17, Brian



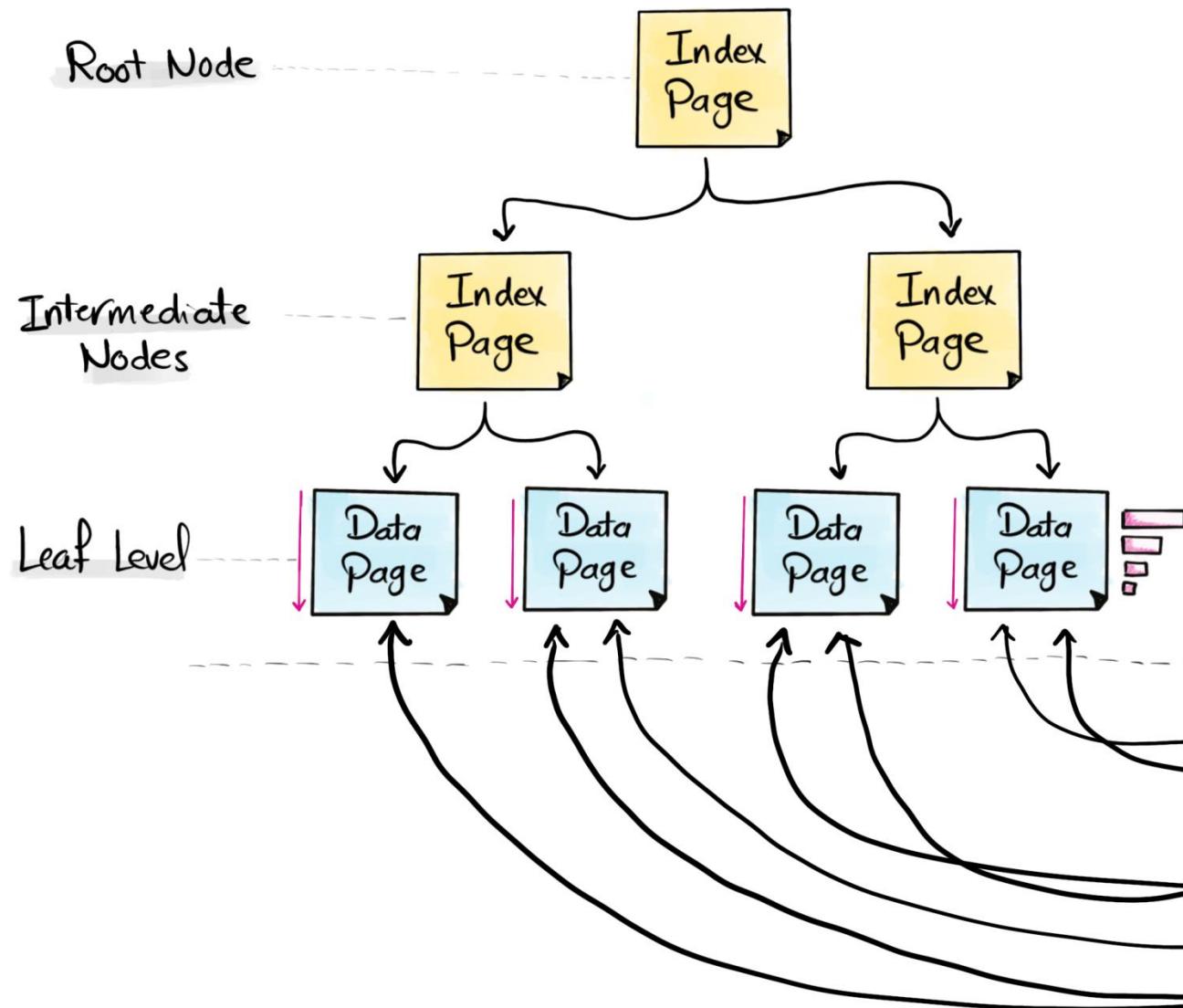
# Clustered INDEX



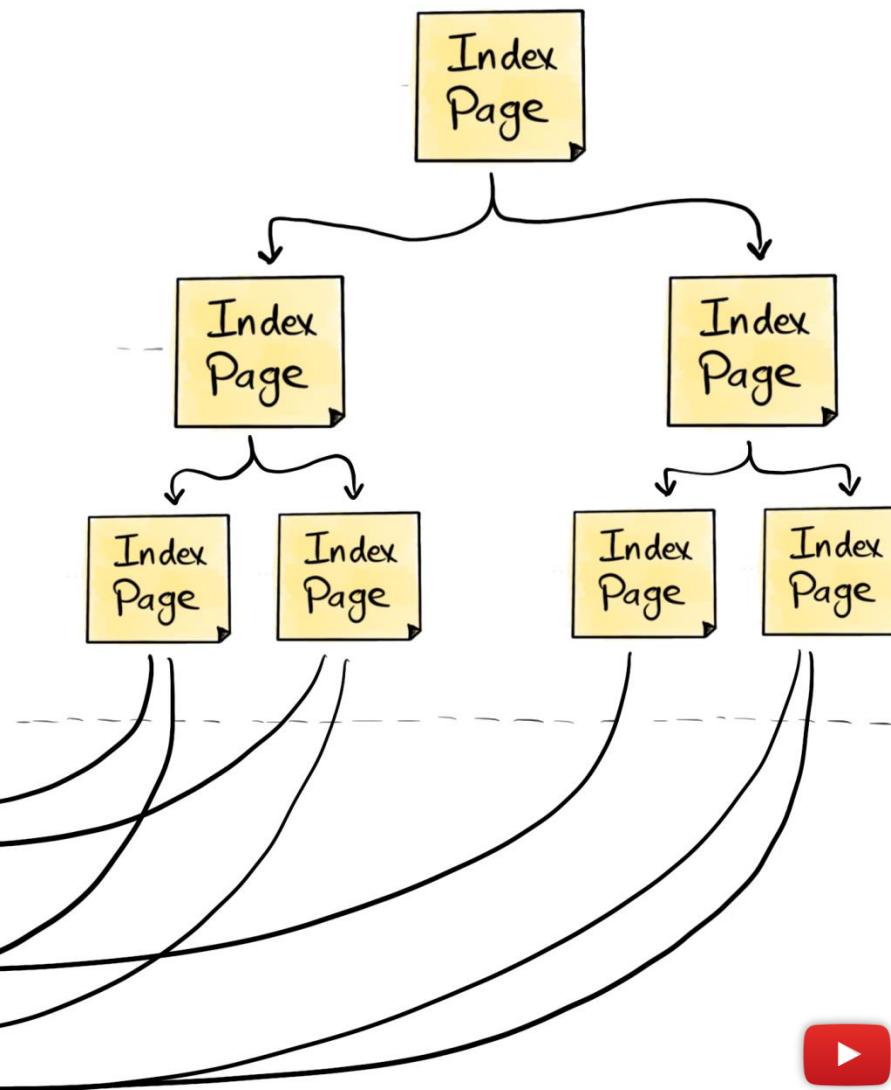
# Non-Clustered Index



## Clustered Index



## Non-Clustered Index



## Clustered Index

## Non-Clustered Index

Definition

Physically sorts and stores rows

Separate structure with pointers to the data

Number of indexes

**One** Index per Table

**Multiple** indexes are allowed

Read Performance

**Faster**

**Slower**

Write Performance

**Slower**, due to potential data row reordering

**Faster**, since physical data order is unaffected

Storage Efficiency

**More storage-efficient**

Requires **additional** storage space

Use Case

- Unique Column
- Not frequently modified Column
- Improve range query performance

- Columns frequently used in search conditions and joins
- Exact match queries

# Index Syntax

Default is  
**NONCLUSTERED**

```
CREATE [CLUSTERED | NONCLUSTERED] INDEX index_name ON table_name (column1, column2, ...)
```

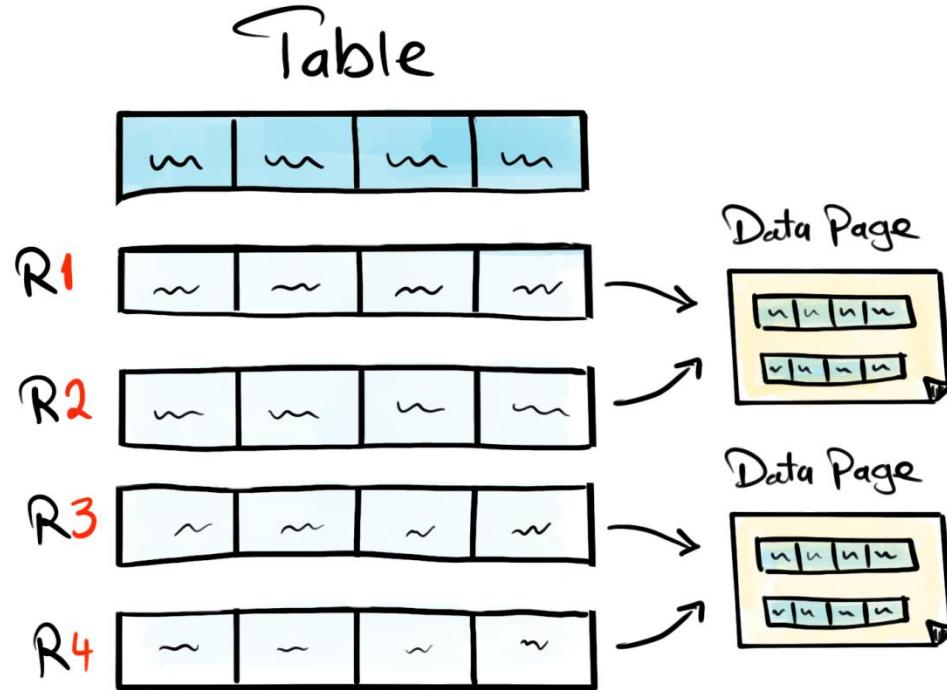
```
CREATE CLUSTERED INDEX IX_Customers_ID ON Customers (ID)
```

```
CREATE NONCLUSTERED INDEX IX_Customers_City ON Customers (City)
```

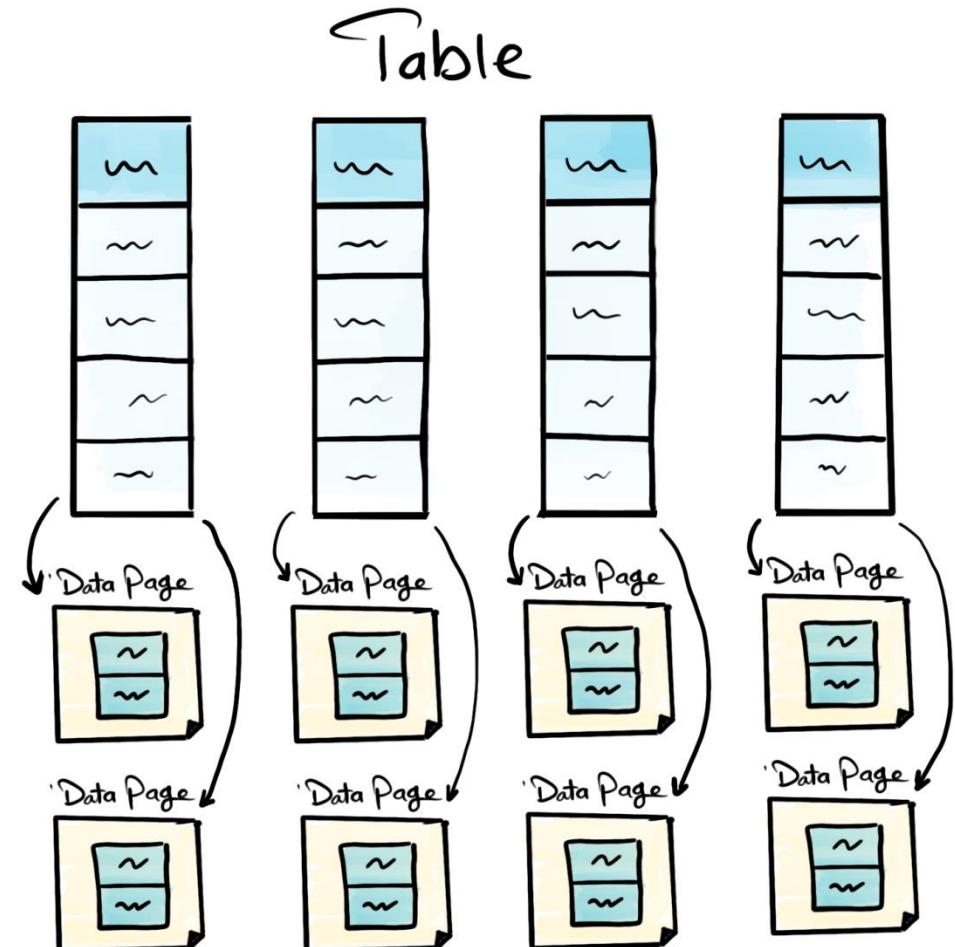
```
CREATE INDEX IX_Customers_Name ON Customers (LastName ASC, FirstName DESC)
```

NONCLUSTERED

## Rowstore Index

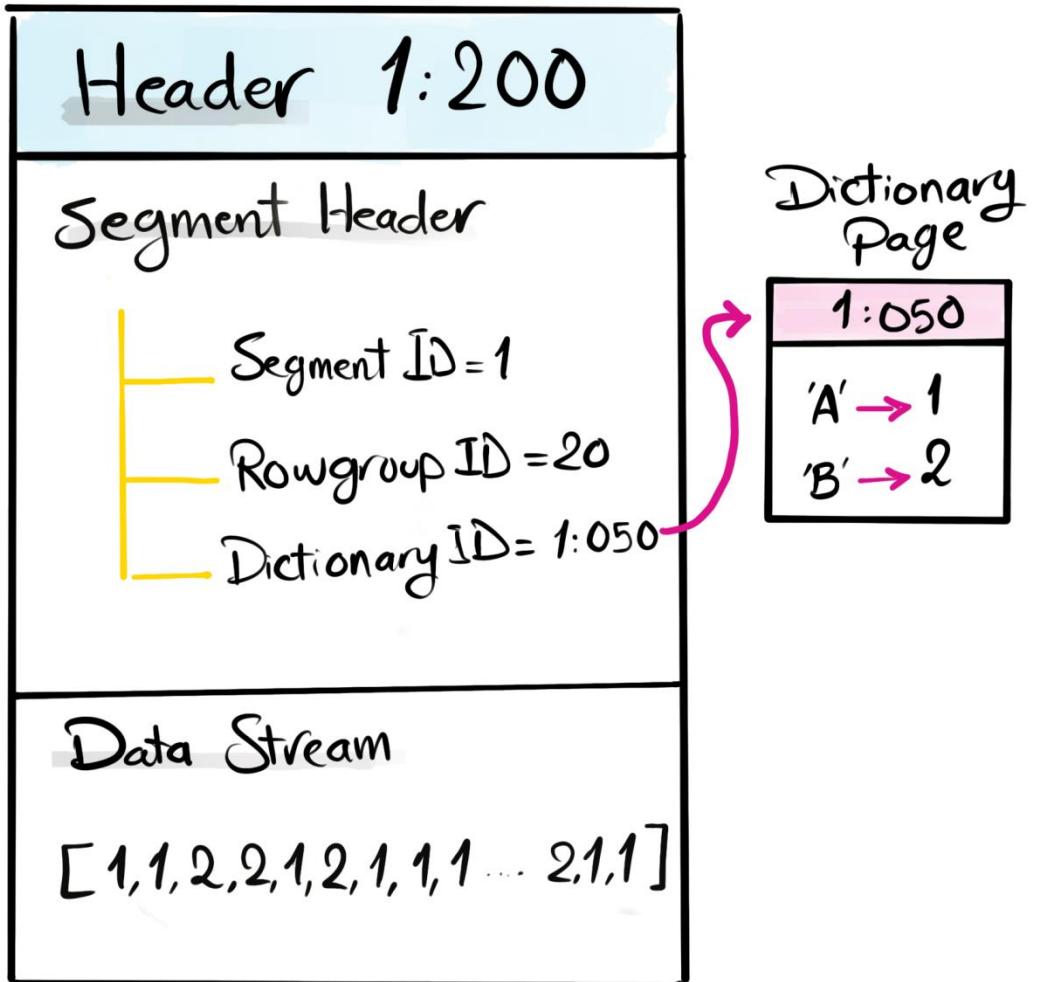


## Columnstore Index



# Column Store

LoB Page



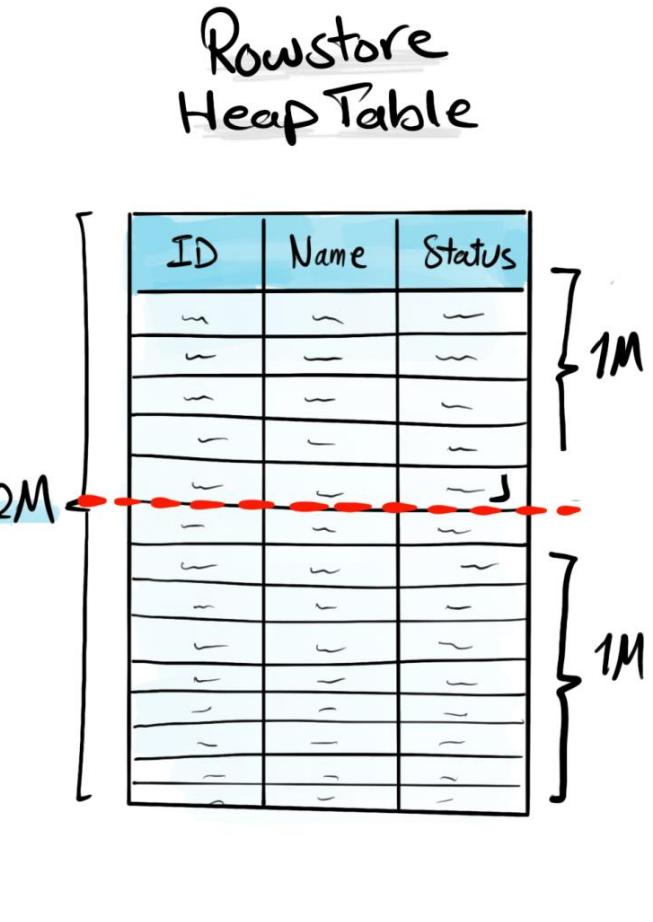
# Row Store

Data Page

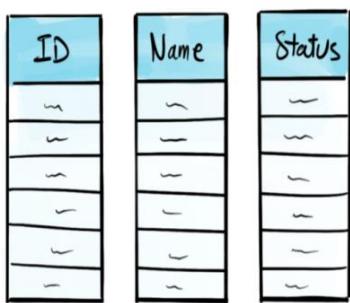
0	Header 1:150	
96	Row 1	1, Bob, USA
118	Row 2	2, Anna, Germany
140	Row 3	3, Lisa, Italy
:		:
FREE SPACE		3 2 1
Offset		140 118 96

## Columnstore Process

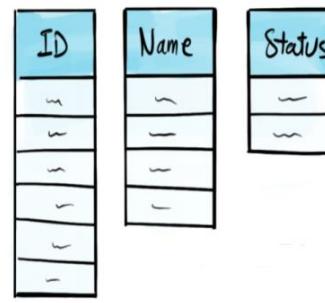
## #1 Row Groups



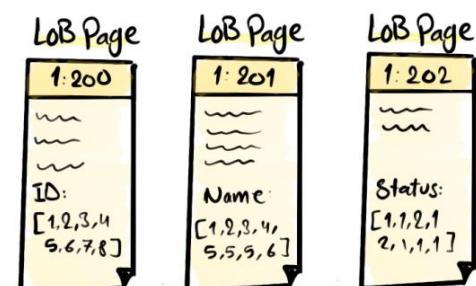
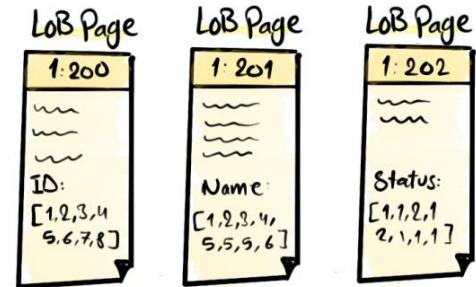
## #2 Column Segment

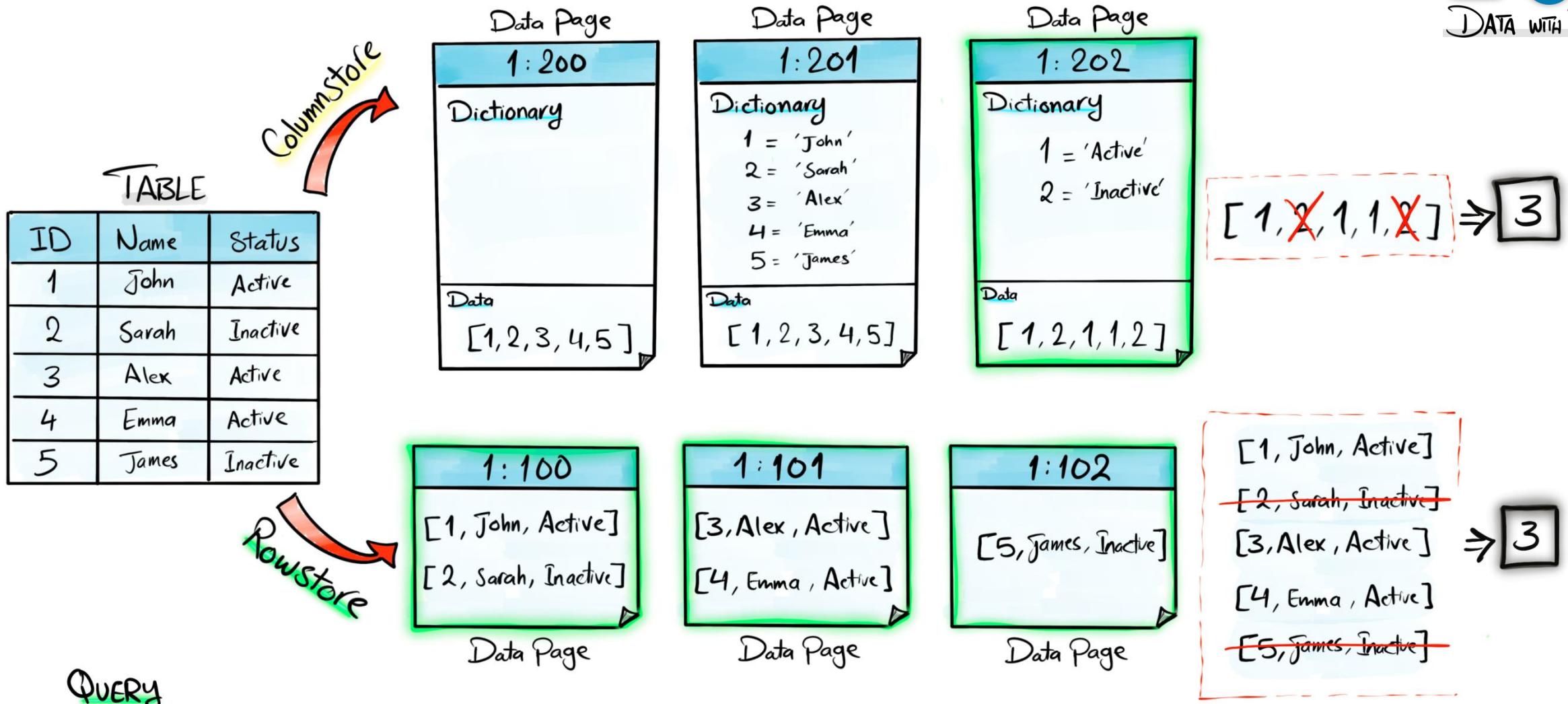


## #3 Compression



## #4 Store





## Rowstore Index

## Columnstore Index

### Definition

Organizes and stores data **row by row**

### Storage Efficiency

**Less efficient** in storage

Organizes and stores data **column by column**

**Highly efficient** with **Compression**

### Read/Write Optimization

Fair speed for read & write operations

**Fast** read performance  
**Slow** write performance

### I/O Efficiency

**Lower** (retrieves **all columns**)

**Higher** (retrieves **specific columns**)

### Best for ..

#### **OLTP (Transactional)**

commerce, banking, Financial systems, order processing

#### **OLAP (Analytical)**

Data Warehouse, Business intelligence, Reporting, Analytics

### Use Case

- High-frequency transaction applications
- Quick access to complete records

- Big Data Analytics
- Scanning of large datasets
- Fast aggregation

## Columnstore Index

Default is  
ROWSTORE

CRE

## Unique Index

Rowstore

Ensures no duplicate values exist in specific column.

Columnstore

### Benefits

1. Enforce uniqueness
2. Slightly increase query performance

Customers (Country)

NOT ALLOWED  
TO USE  
COLUMNS

Rules

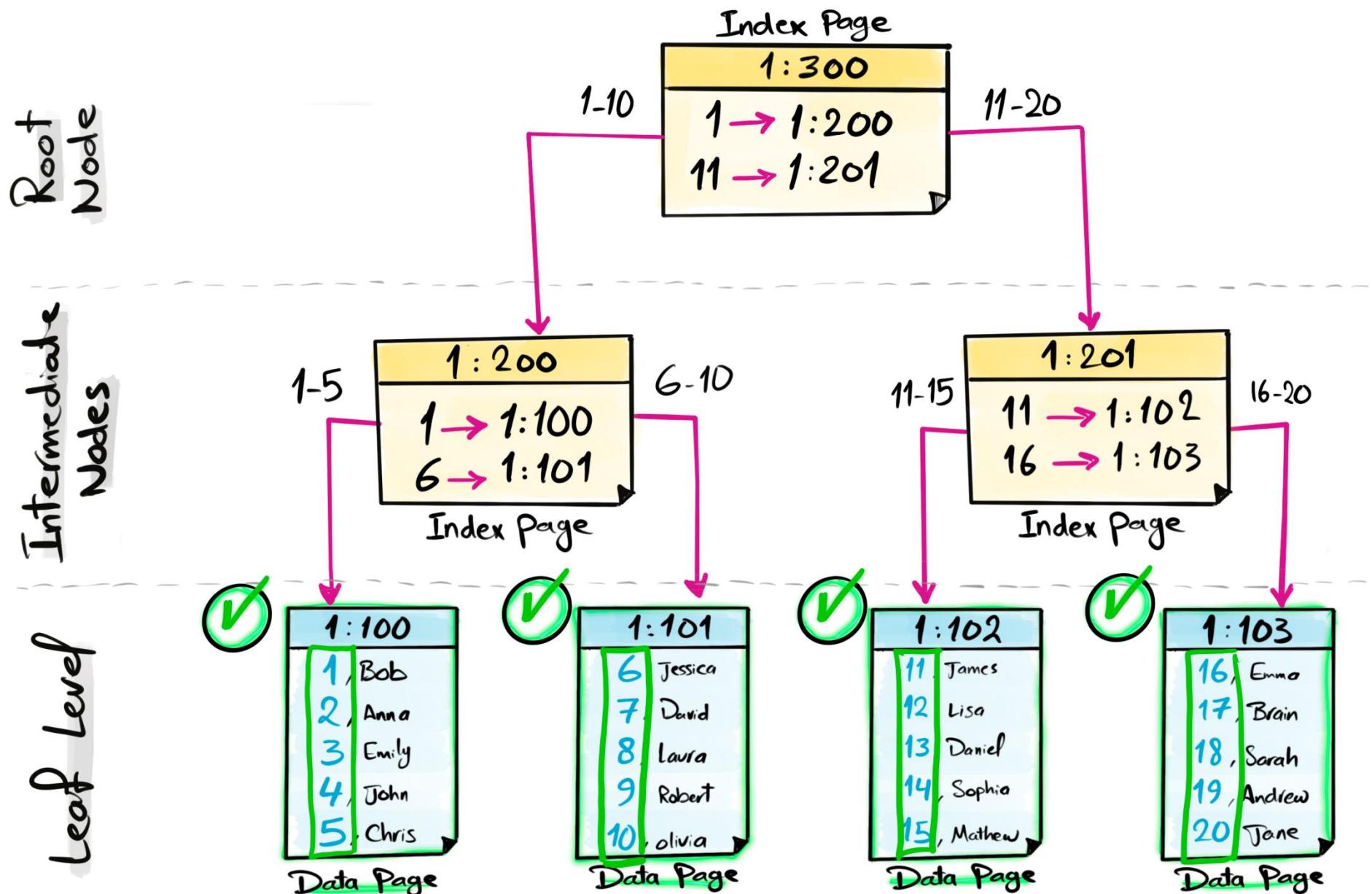
- You **can't** specify specific columns in Clustered Index Columnstore

# Unique Index

Ensures no duplicate values exist in specific column.

## Benefits

1. Enforce uniqueness
  2. Slightly increase query performance
-



# Unique Index

Default is  
**NOT Unique**



```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] [COLUMNSTORE] INDEX index_name  
ON table_name (column1, column2, ...)
```

Index Allows  
Duplicates

```
CREATE INDEX IX_Customers_Email ON Customers (Email)
```

Duplicates  
are not  
allowed

```
CREATE UNIQUE INDEX IX_Customers_Email ON Customers (Email)
```

# Filtered Index

An index that includes only rows  
meeting the specified conditions

## Benefits

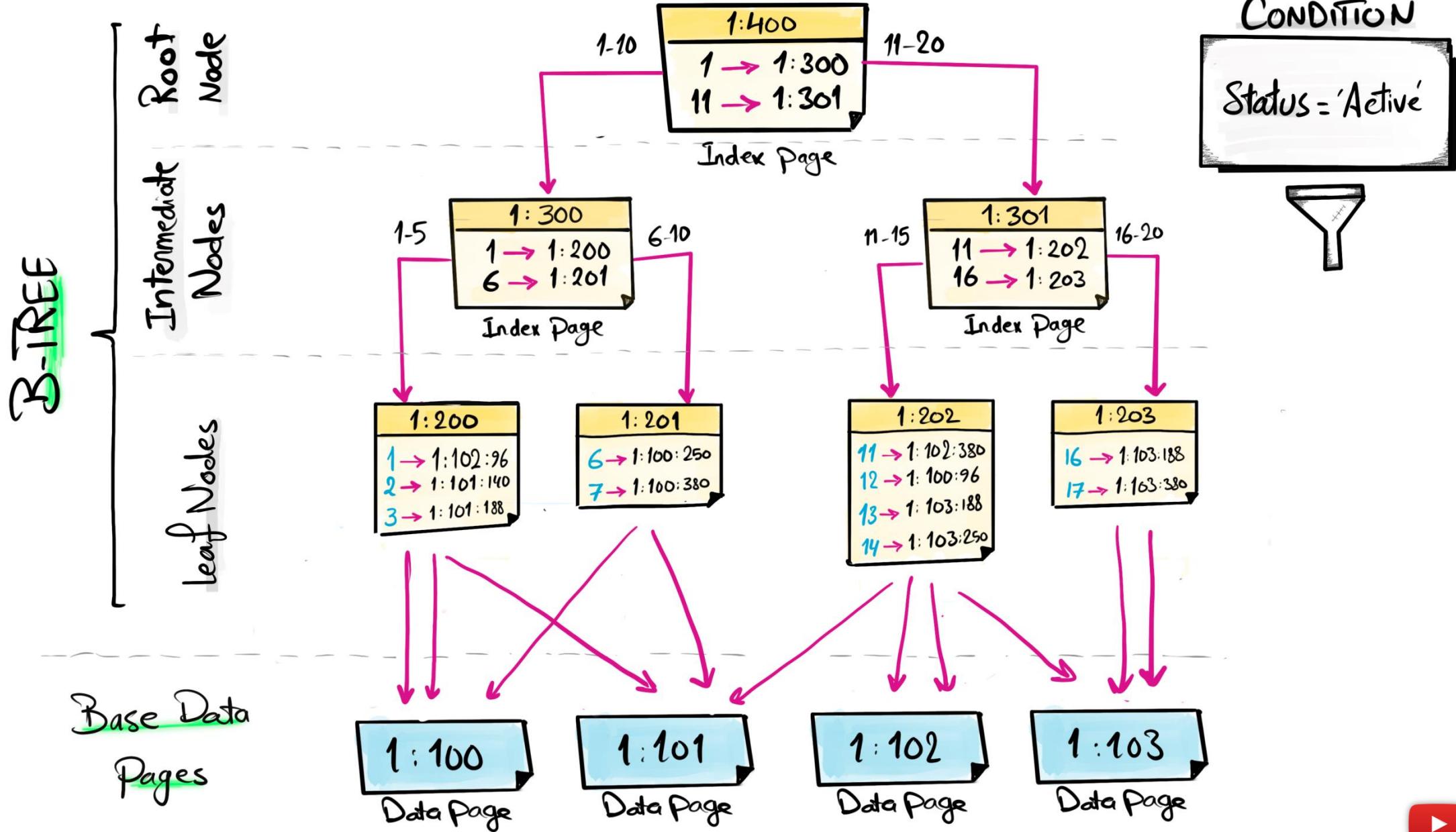
- Targeted Optimization
- Reduce storage: Less data in the index

## Filtered Index

```
CREATE [UNIQUE] [NONCLUSTERED] INDEX index_name  
ON table_name (column1, column2, ...)  
WHERE [Condition]
```

### Rules

- You **cannot** create a filtered index on a **clustered index**.
- You **cannot** create a filtered index on a **columnstore index**.



## When To Use

### HEAP

Fast **Inserts**  
(For Staging Tables)

### MAIN

### Clustered Index

For **Primary keys**  
If not, then for date columns

### OLTP

### Columnstore Index

For **Analytical** Queries  
Reduce Size of Large Table

### OLAP

### Non-Clustered Index

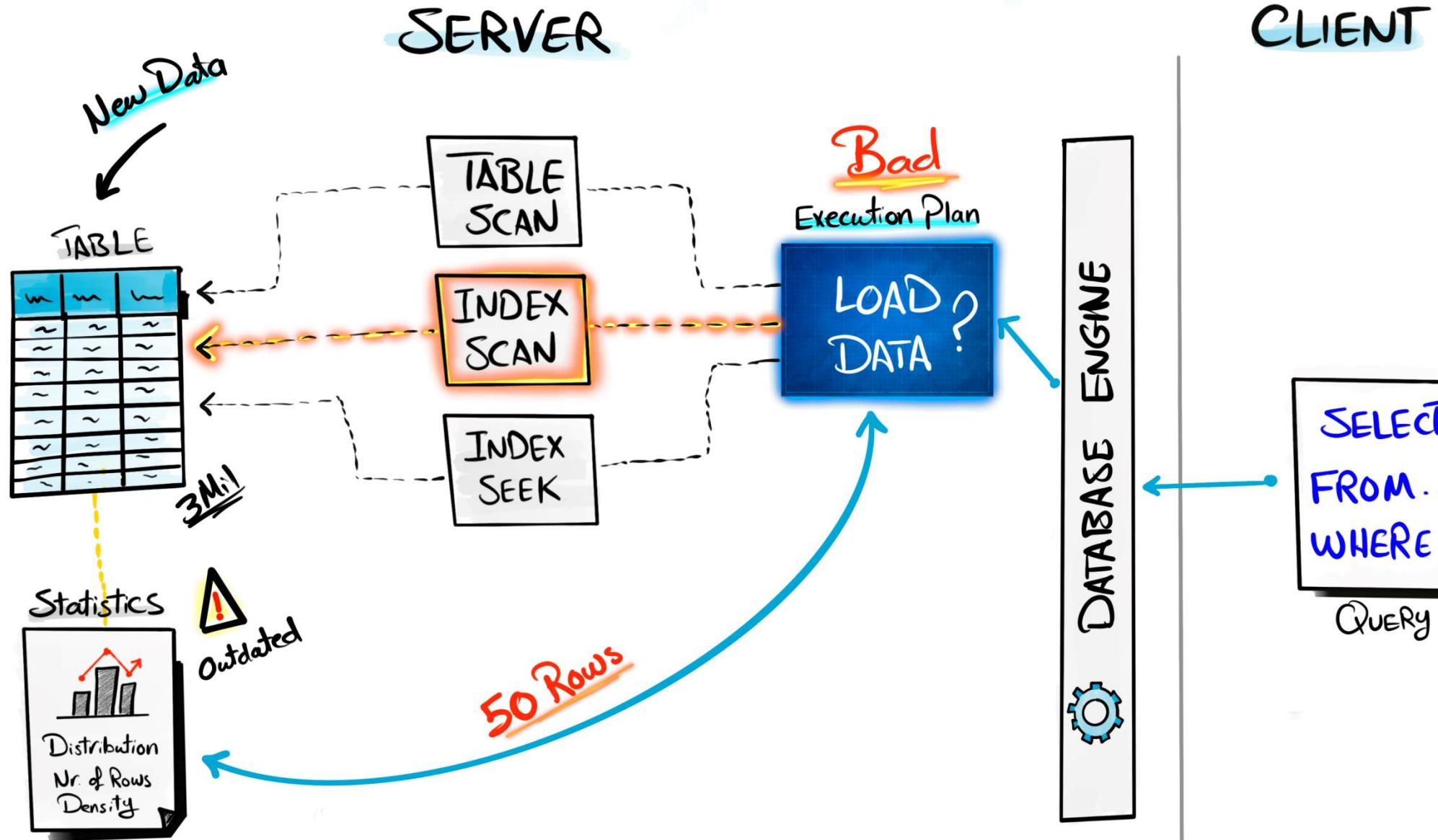
For **non-PK** columns  
(Foreign keys, Joins, and Filters)

### Filtered Index

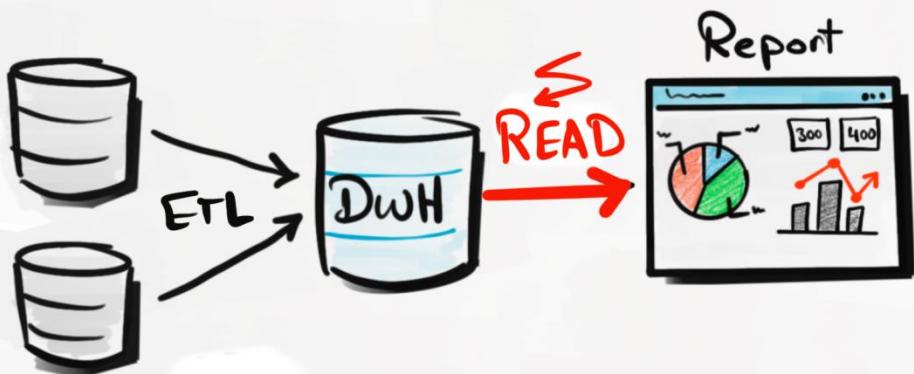
Target **Subset** of Data  
Reduce Size of Index

### Unique Index

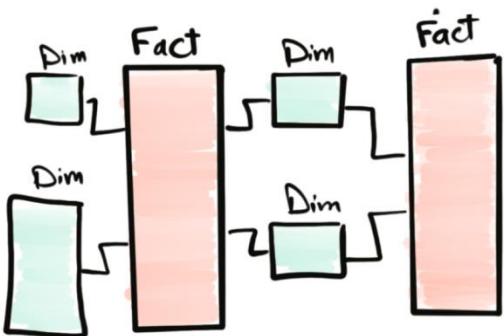
Enforce **Uniqueness**  
Improve Query Speed



# OLAP (Analytical)



## COLUMNSTORE INDEX

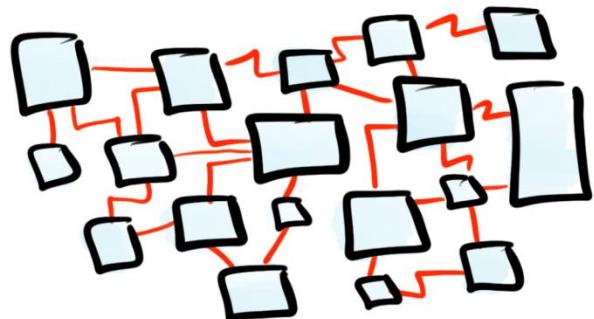


**GOAL** Optimize **READ** Performance

# OLTP (Transaction)



## CLUSTERED INDEX PK



**GOAL** Optimize **WRITE** Performance

# Indexing Strategy

#1

## Initial Indexing Strategy

### OLAP

Optimize **Read** Performance

Switch **Large** frequently used tables into **ColumnStore**

### OLTP

Optimize **Write** Performance

**Clustered Index**  
Primary Keys

#2

## Usage Patterns Indexing

1

Identify **frequently** used **Tables & Columns**

2

Choose **Right** Index

3

**Test** Index

#3

## Scenario-Based Indexing

1

Identify **Slow** Queries

2

Check **Execution Plan**

3

Choose **Right** Index

4

(Test) **Compare** Execution Plans

#4

## Monitoring & Maintenance

1

Monitor Index **Usage**

2

Monitor **Missing** Indexes

3

Monitor **Duplicate** Indexes

4

Update **Statistics**

5

Monitor **Fragmentations**



DATA WITH BARAA

# Partitioning

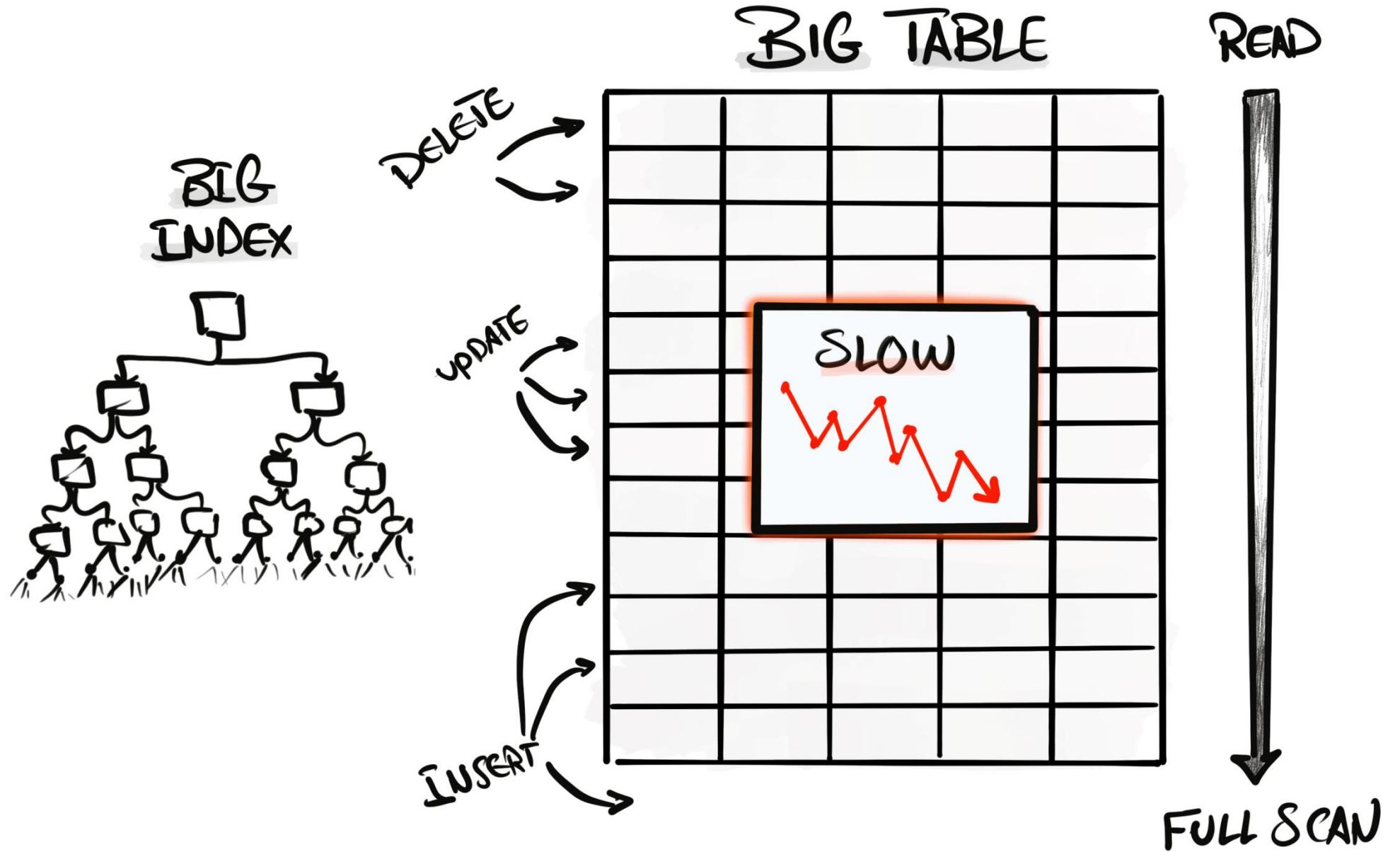
Baraa Khatib Salkini  
YouTube | **DATA WITH BARAA**  
SQL Course | Indexes



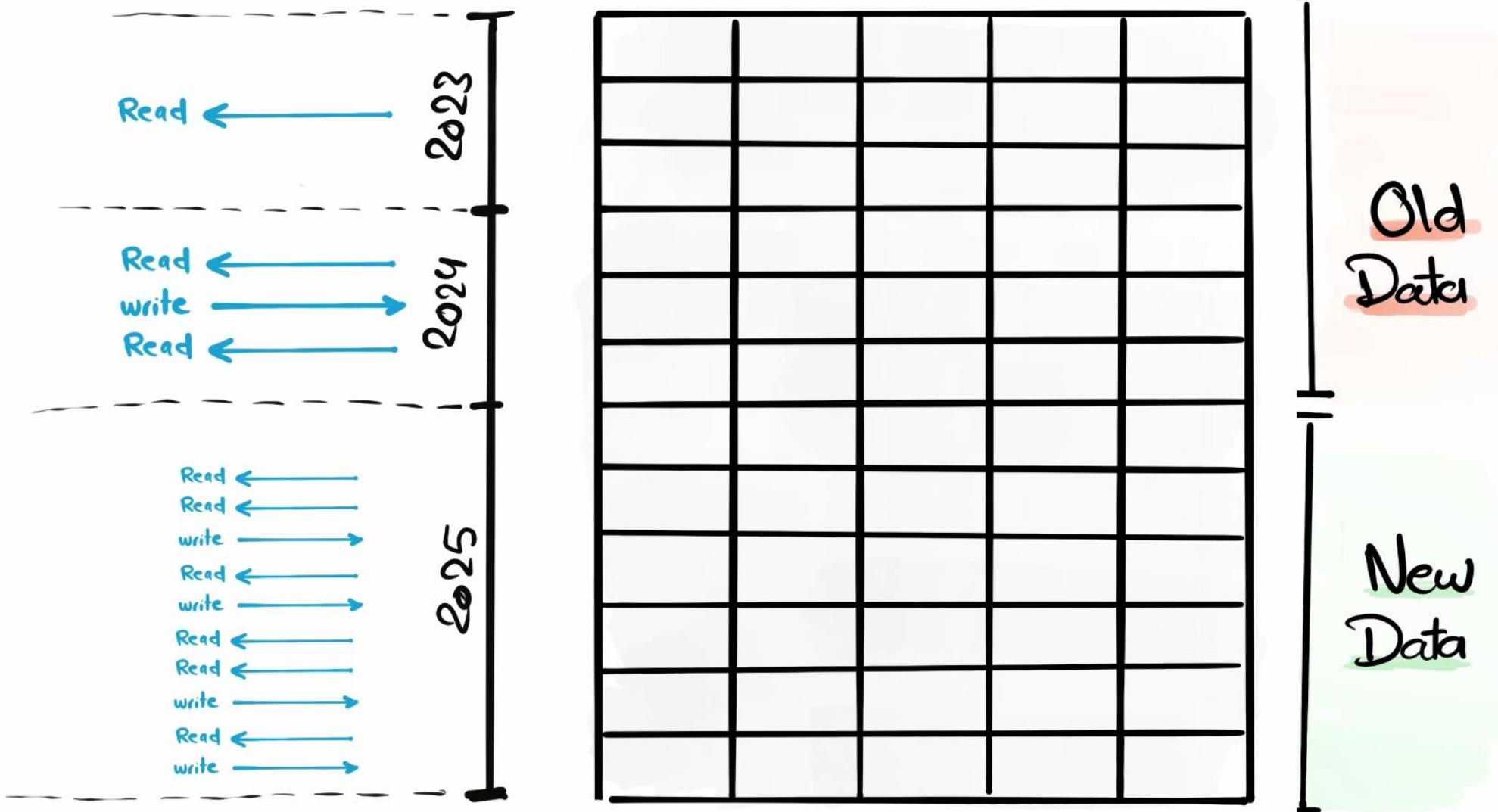
# SQL PARTITIONING

Divides Big Table into Smaller Partitions

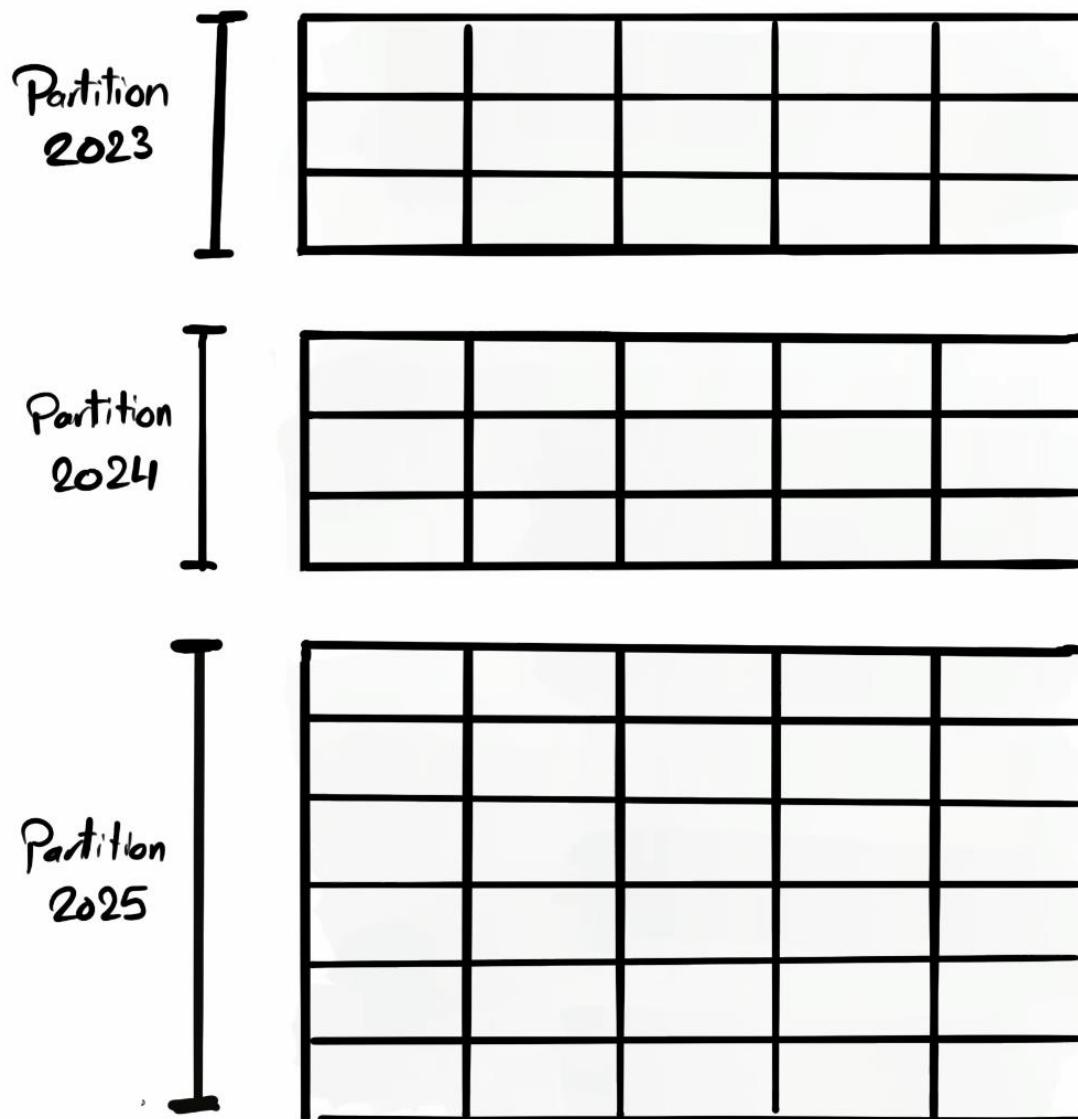
while still being treated as a single logical table.



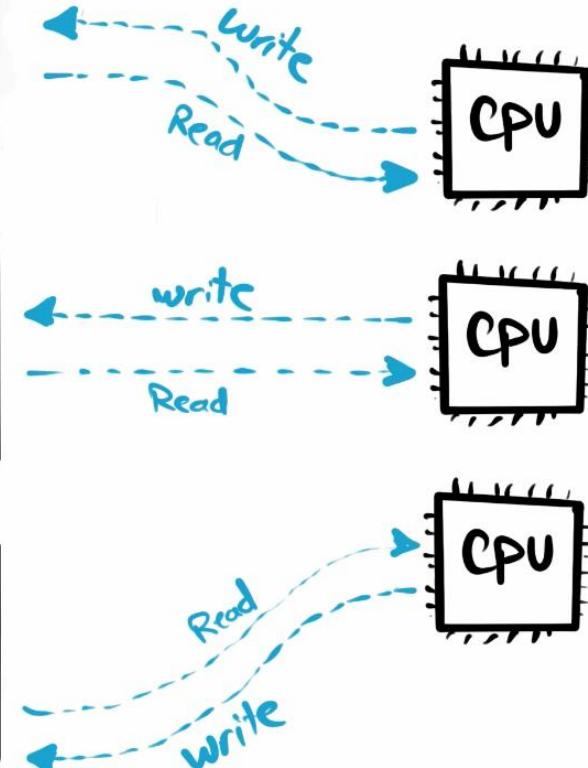
# BIG TABLE



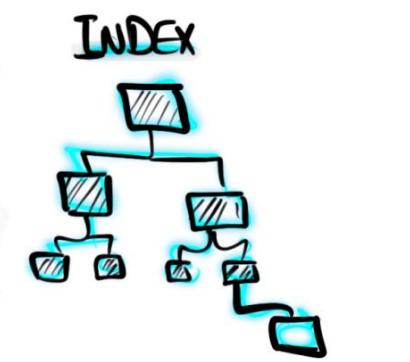
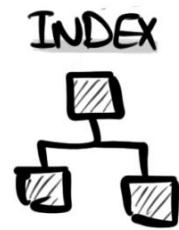
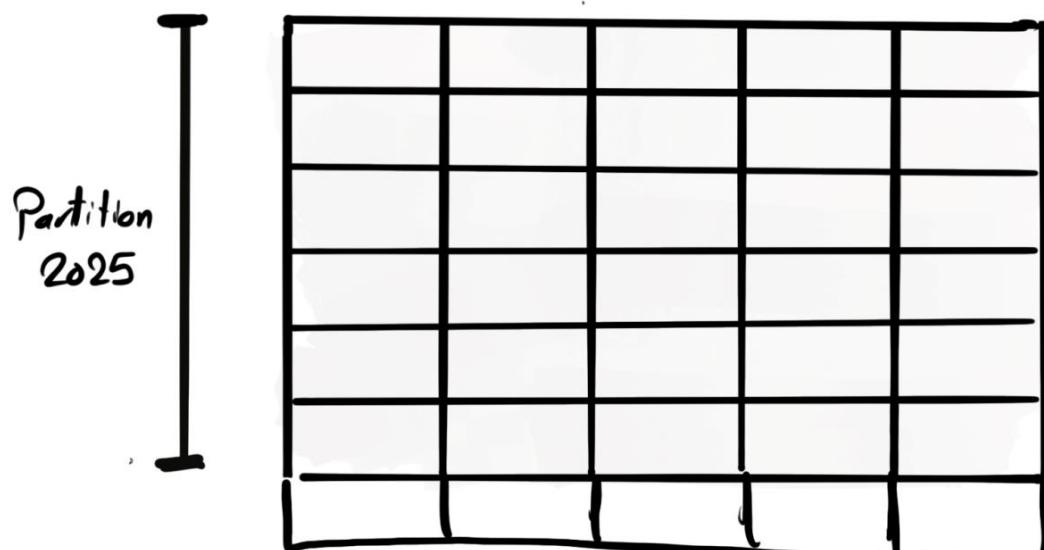
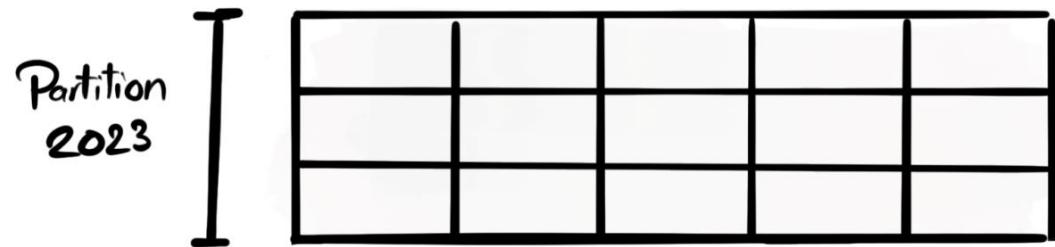
# BIG TABLE



Parallel processing



# BIG TABLE



NEW DATA

## PARTITION FUNCTION

Define the Logic on how to divide  
your data into partitions !

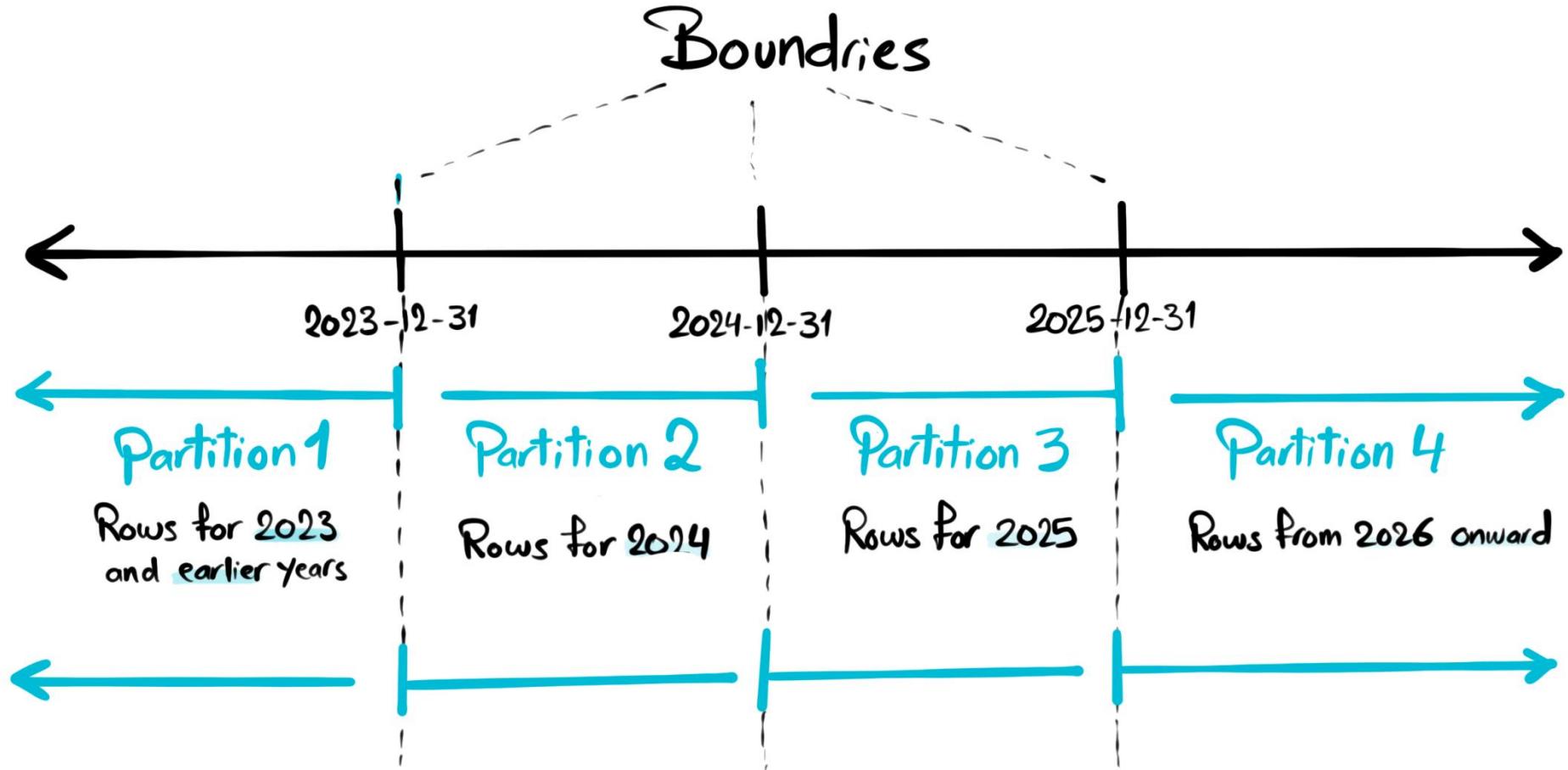
Based on Partition Key Like (Column, Region, ..)

1

# Partition Function

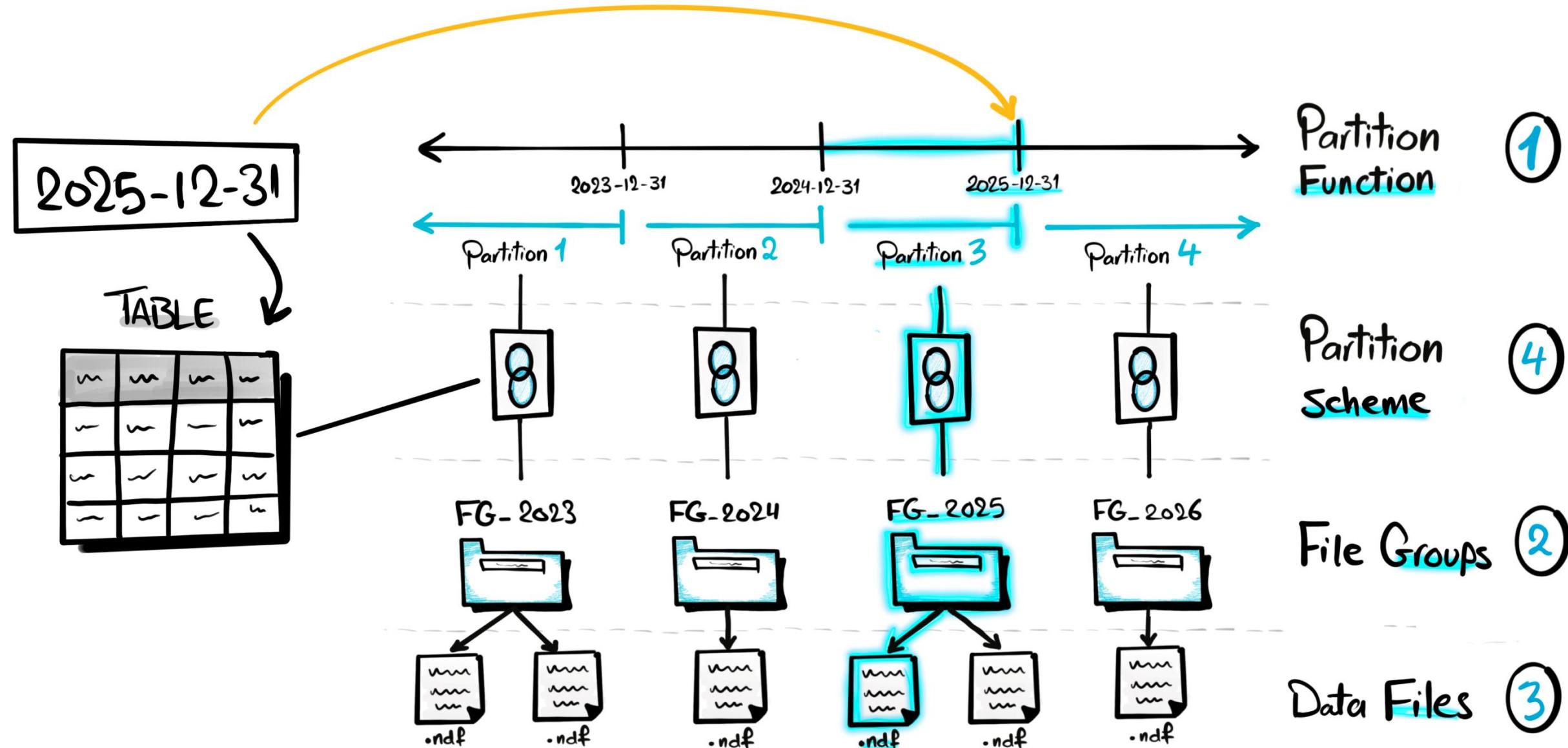
**LEFT Logic**

**RIGHT Logic**

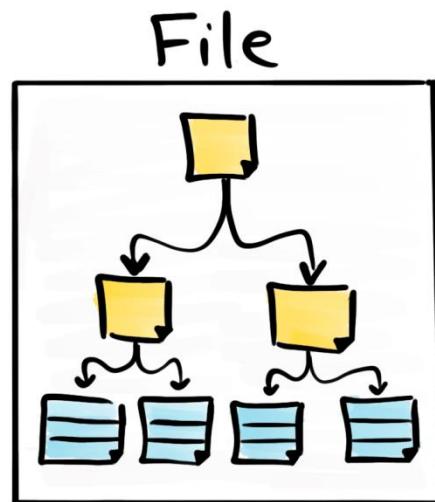


# FILEGROUPS

Logical container of one or more data files  
to help organize partitions.



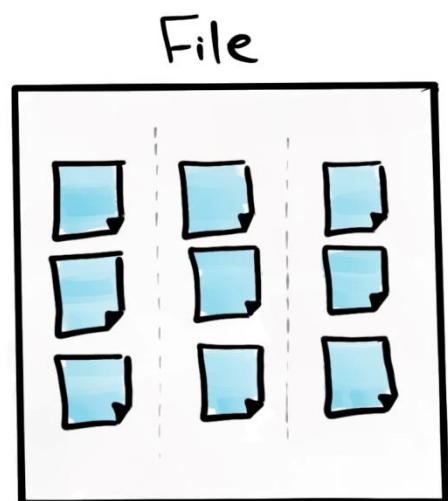
# Indexing



Rowstore  
Index

TABLE

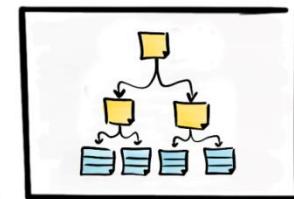
~	~	~
~	~	~
~	~	~
~	~	~



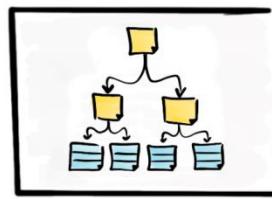
Columnstore  
Index

# Partitioning + Indexing

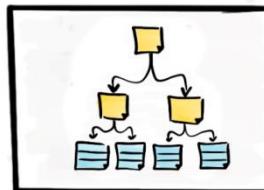
File1



File2



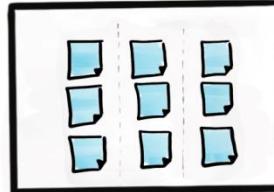
File3



Horizontal  
Partitioning

Vertical  
Partitioning

File1



File2

