



DATA WITH BARAA

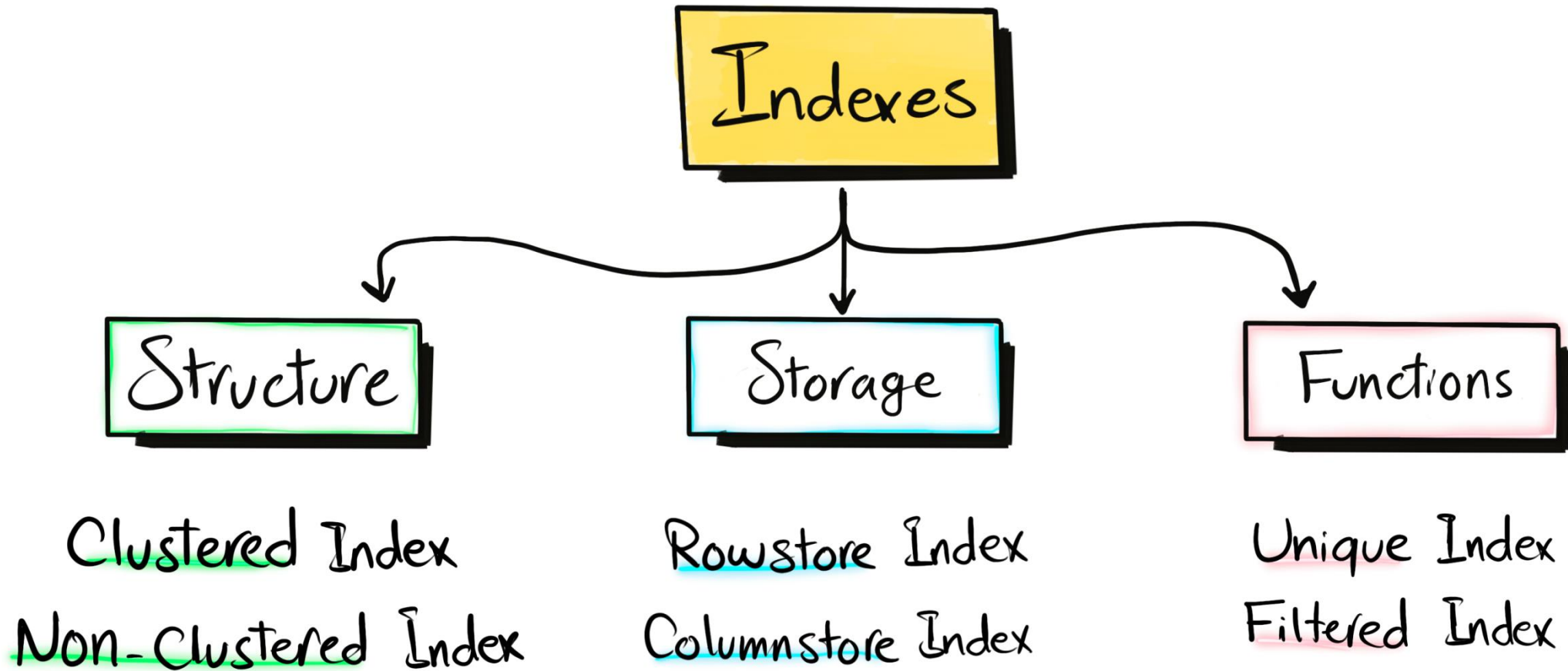
Indexes

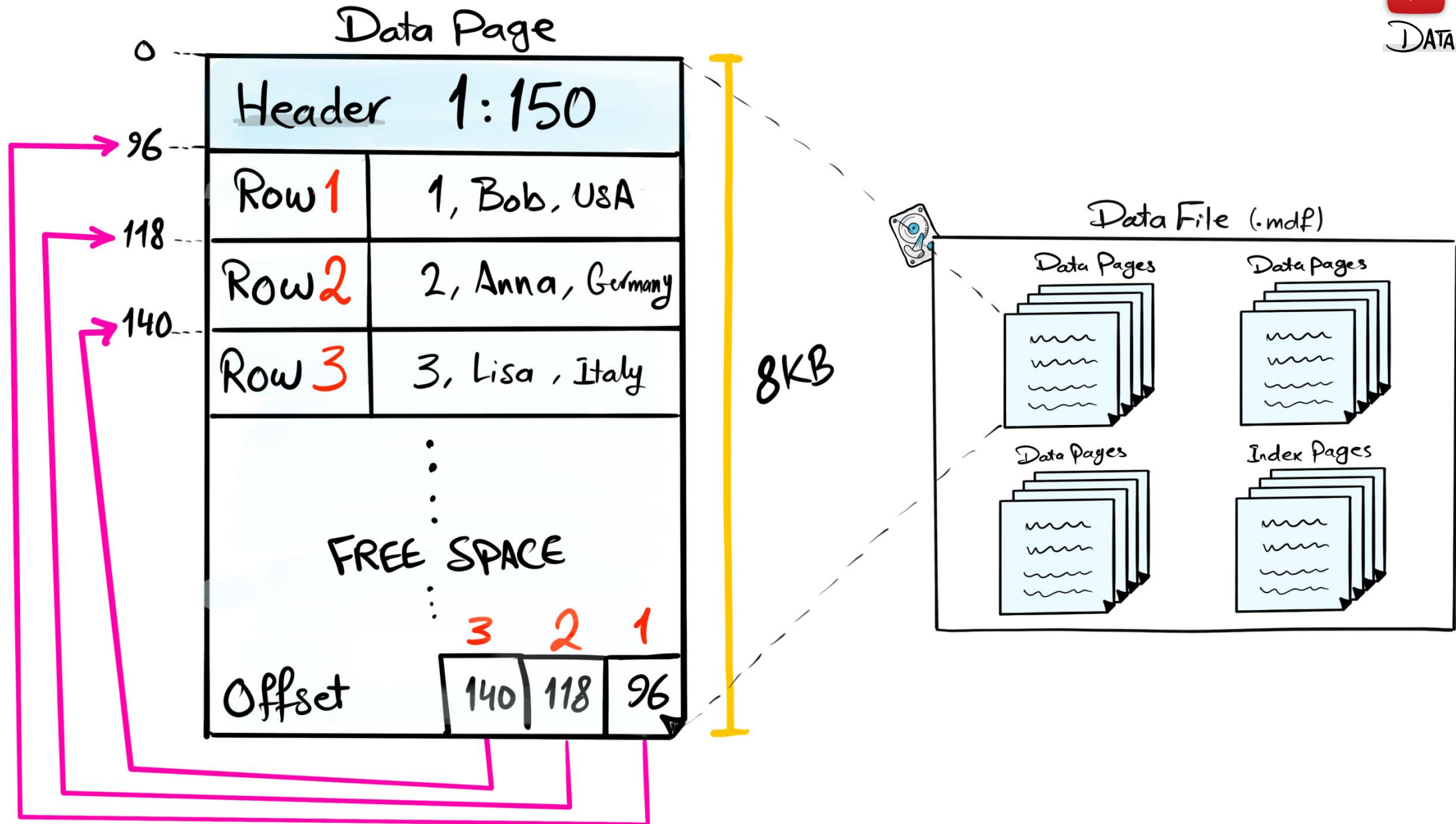
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Indexes



INDEX

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





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

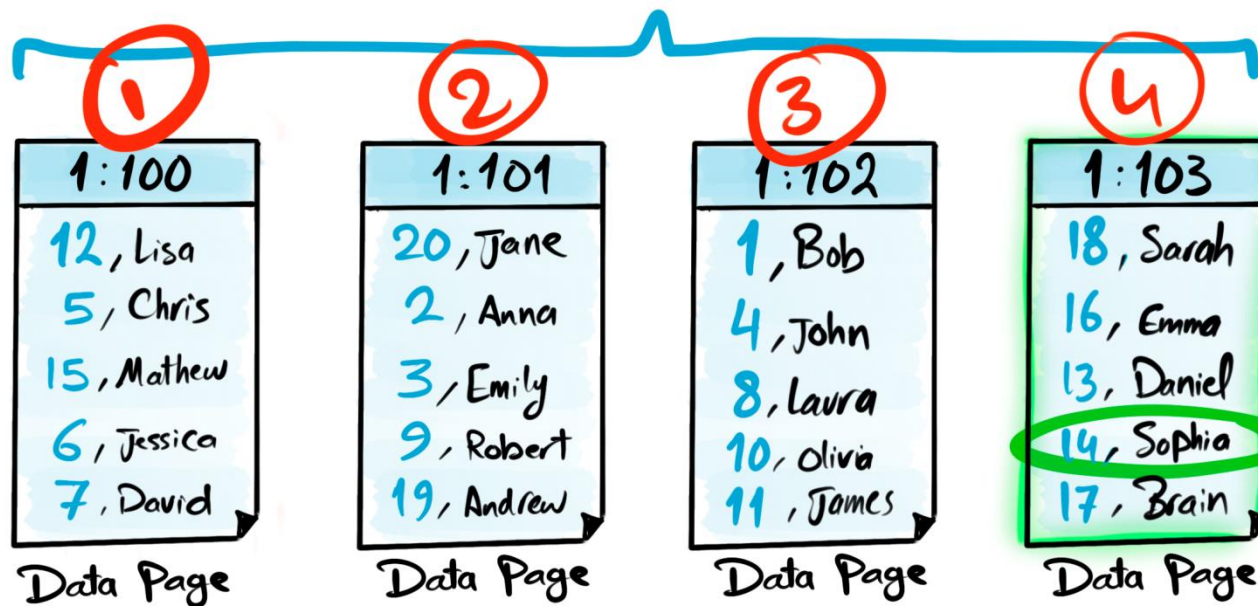
1:103
18, Sarah
16, Emma
13, Daniel
14, Sophia
17, Brian

Data Page



Data File (.mdf)

HEAP



Full Table Scan

CUSTOMERS

ID	Name
1	Bob
2	Anna
⋮	⋮
20	Jane



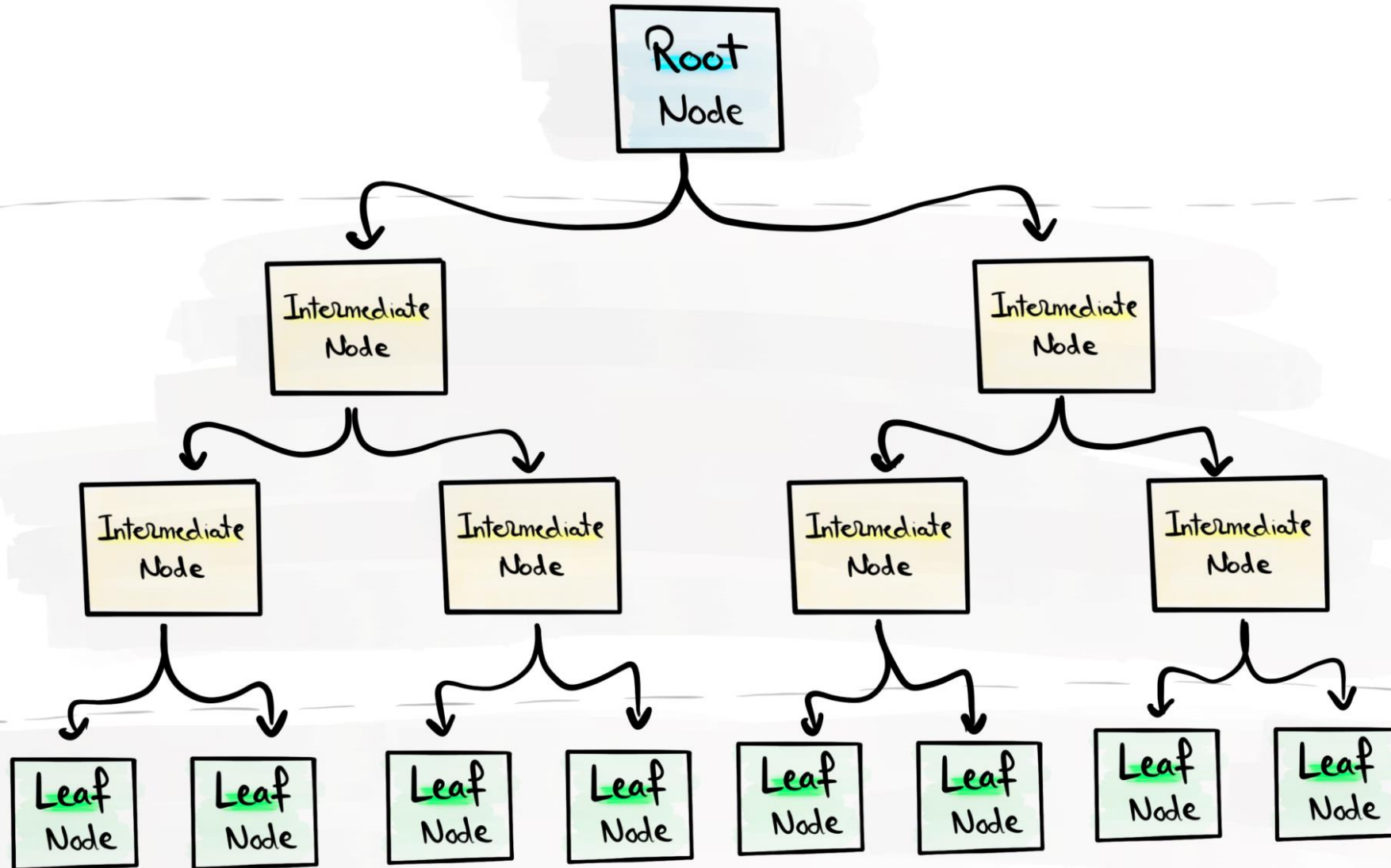
Sophia

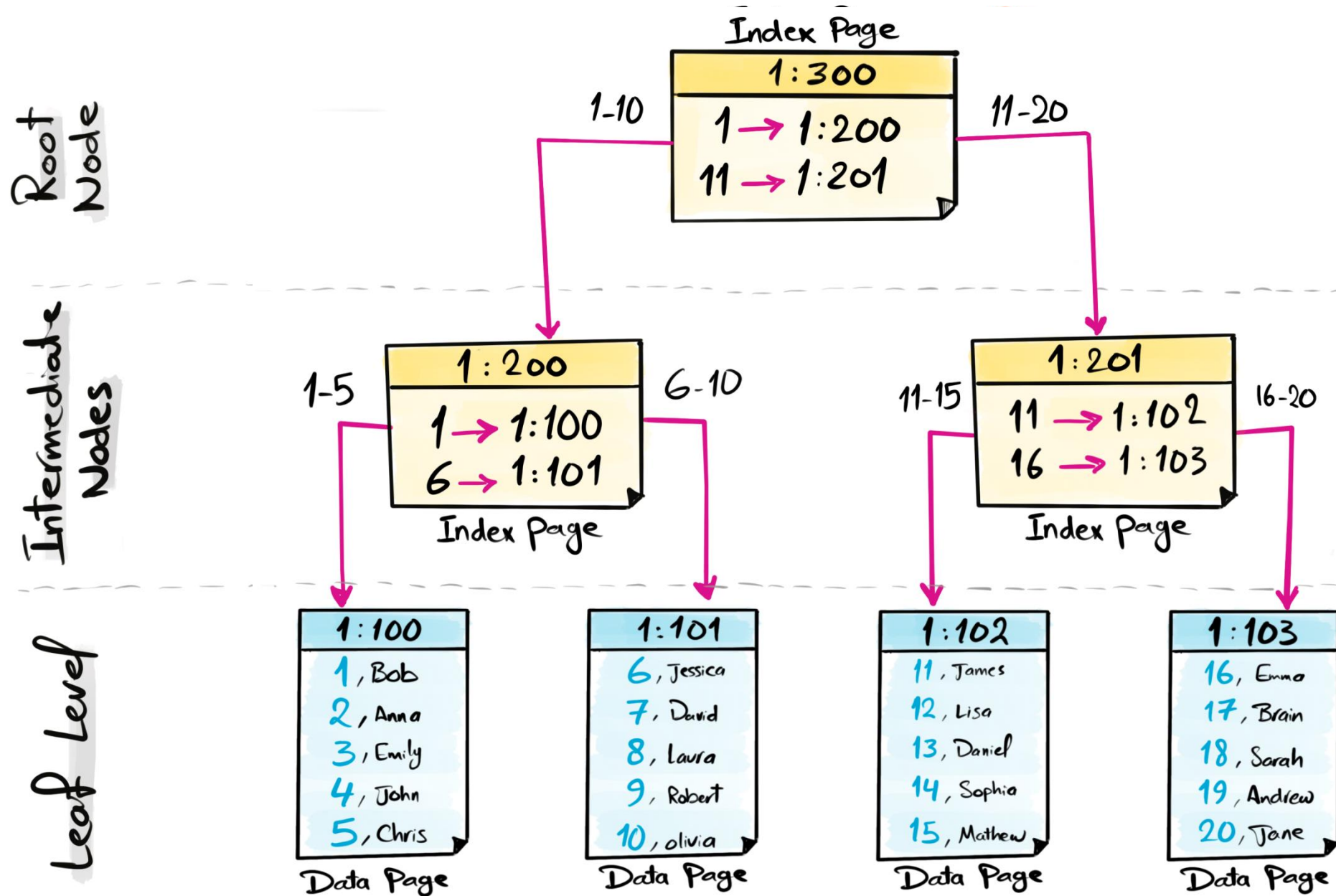
B-TREE

(BALANCE TREE)

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

B-TREE

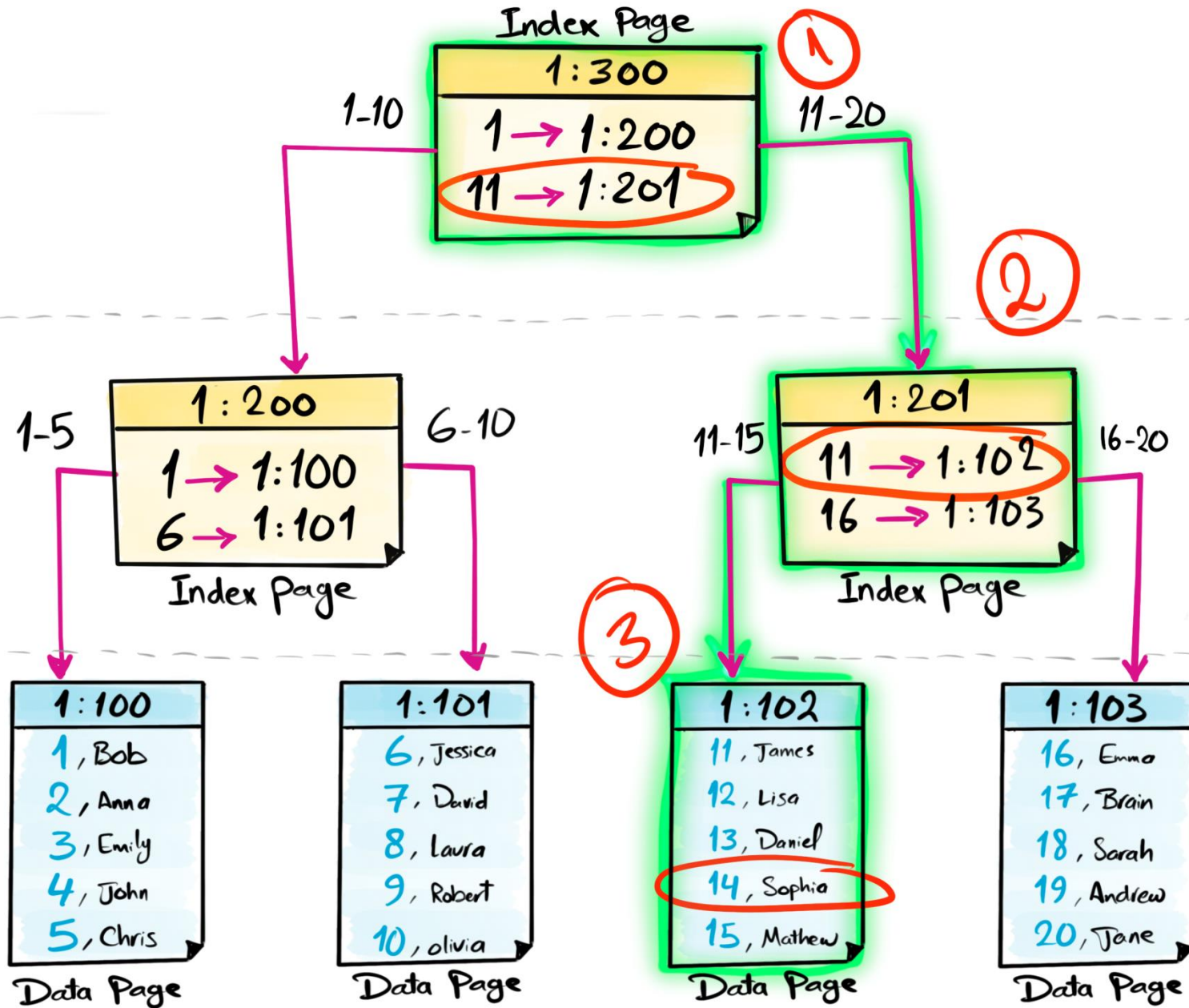




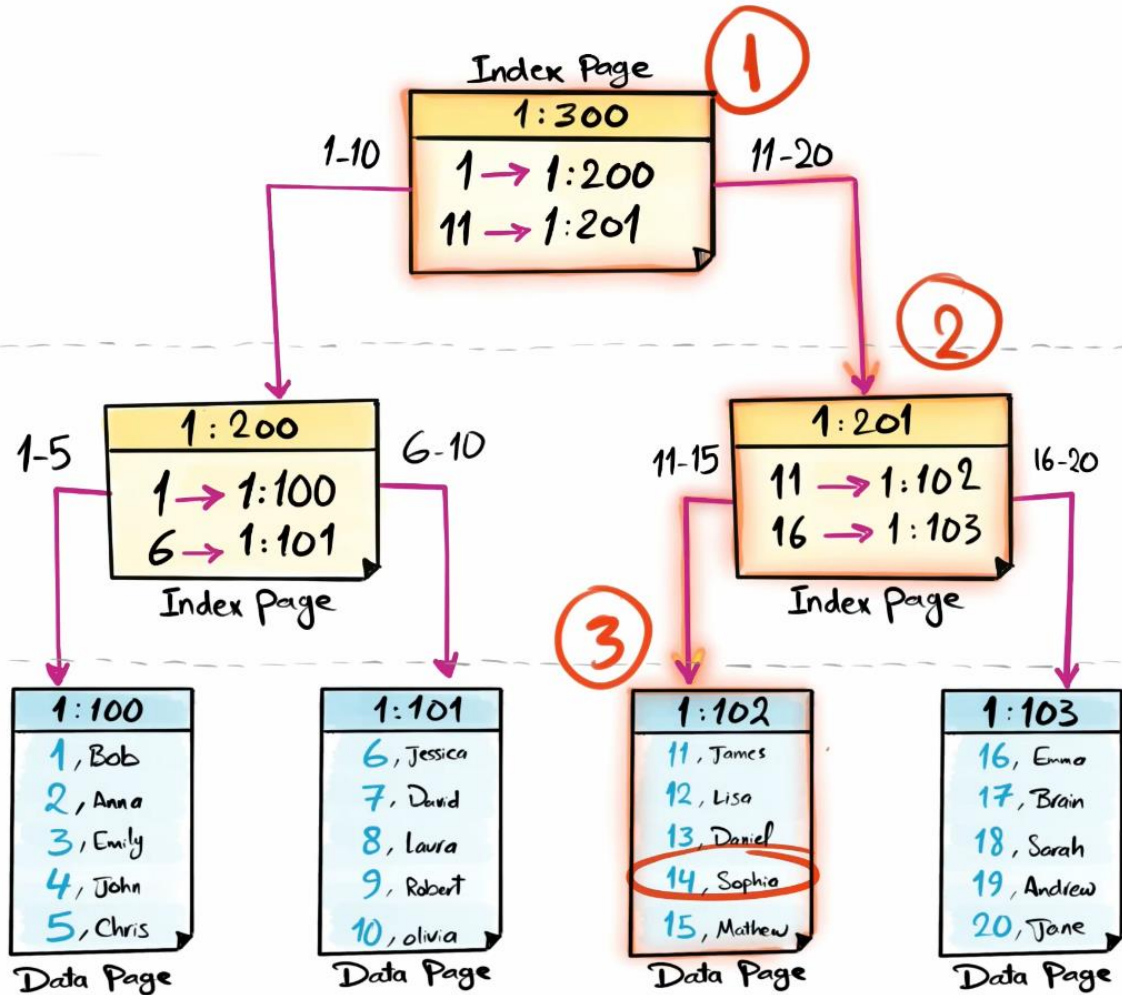
Root Node

Intermediate Nodes

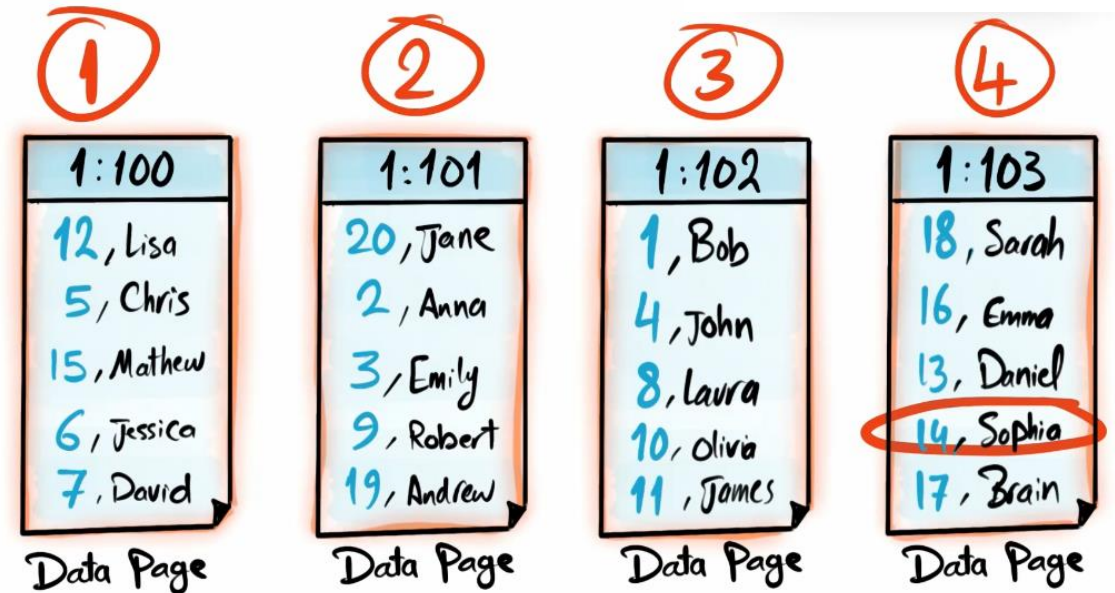
Leaf Level

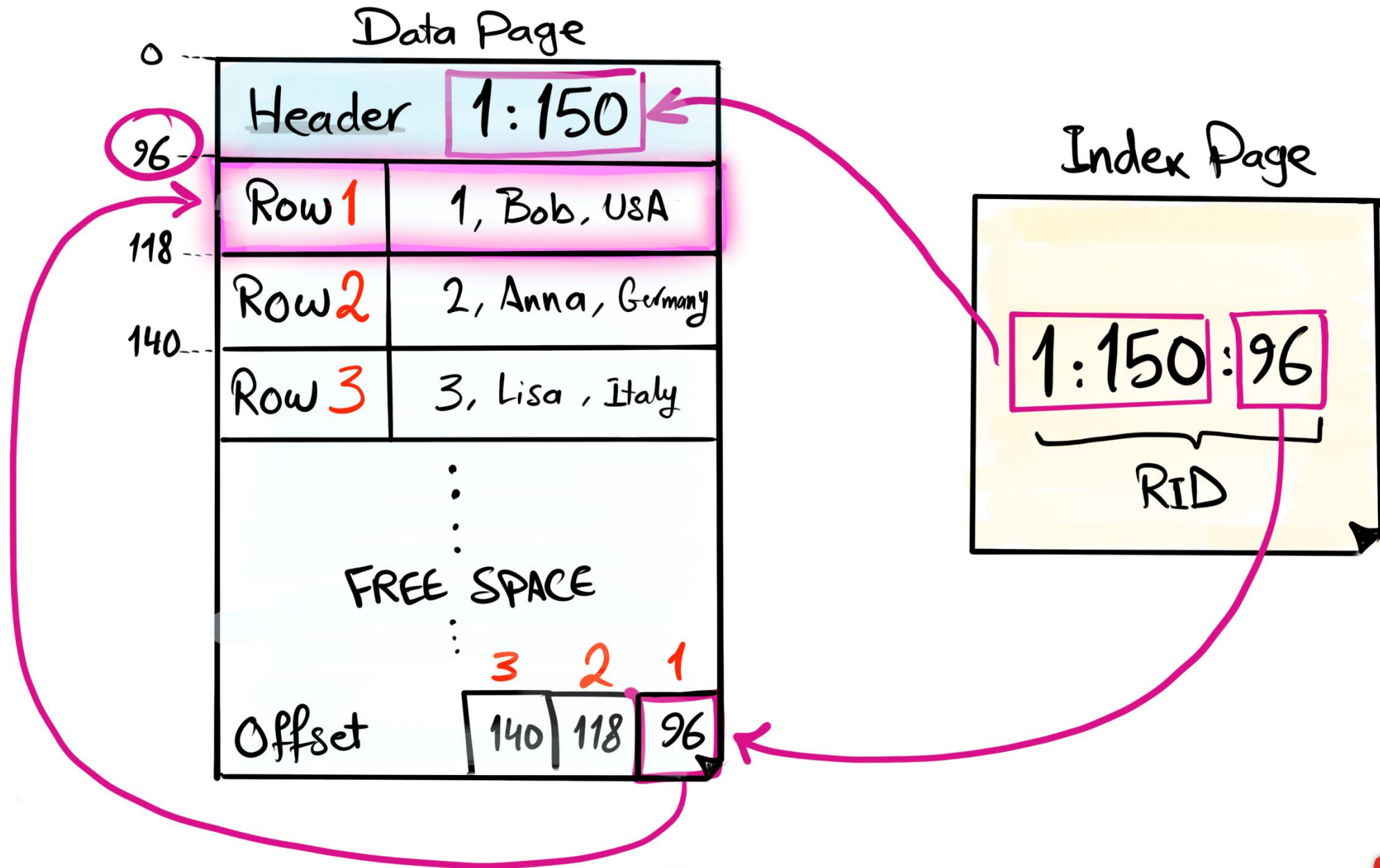


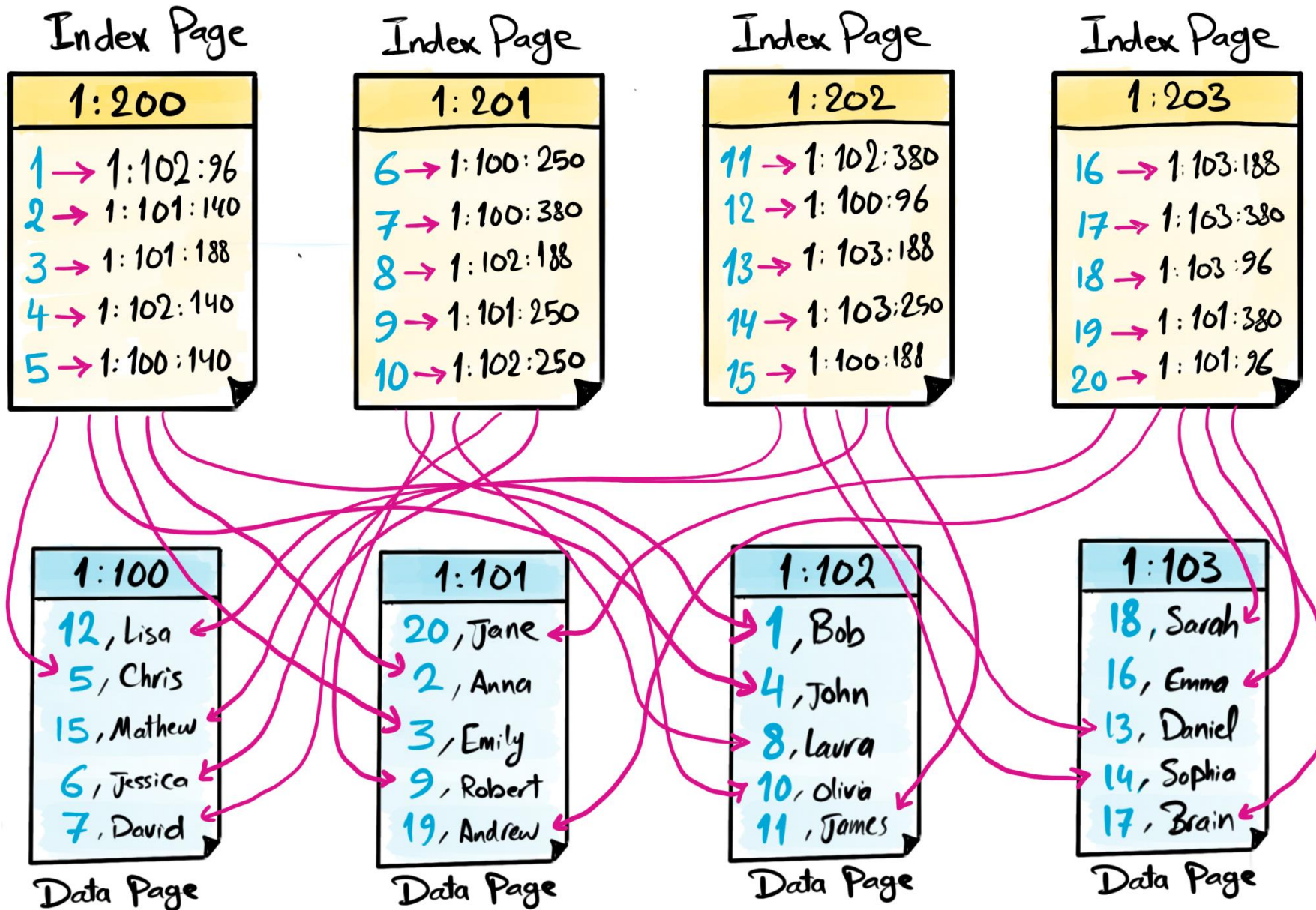
Clustered INDEX

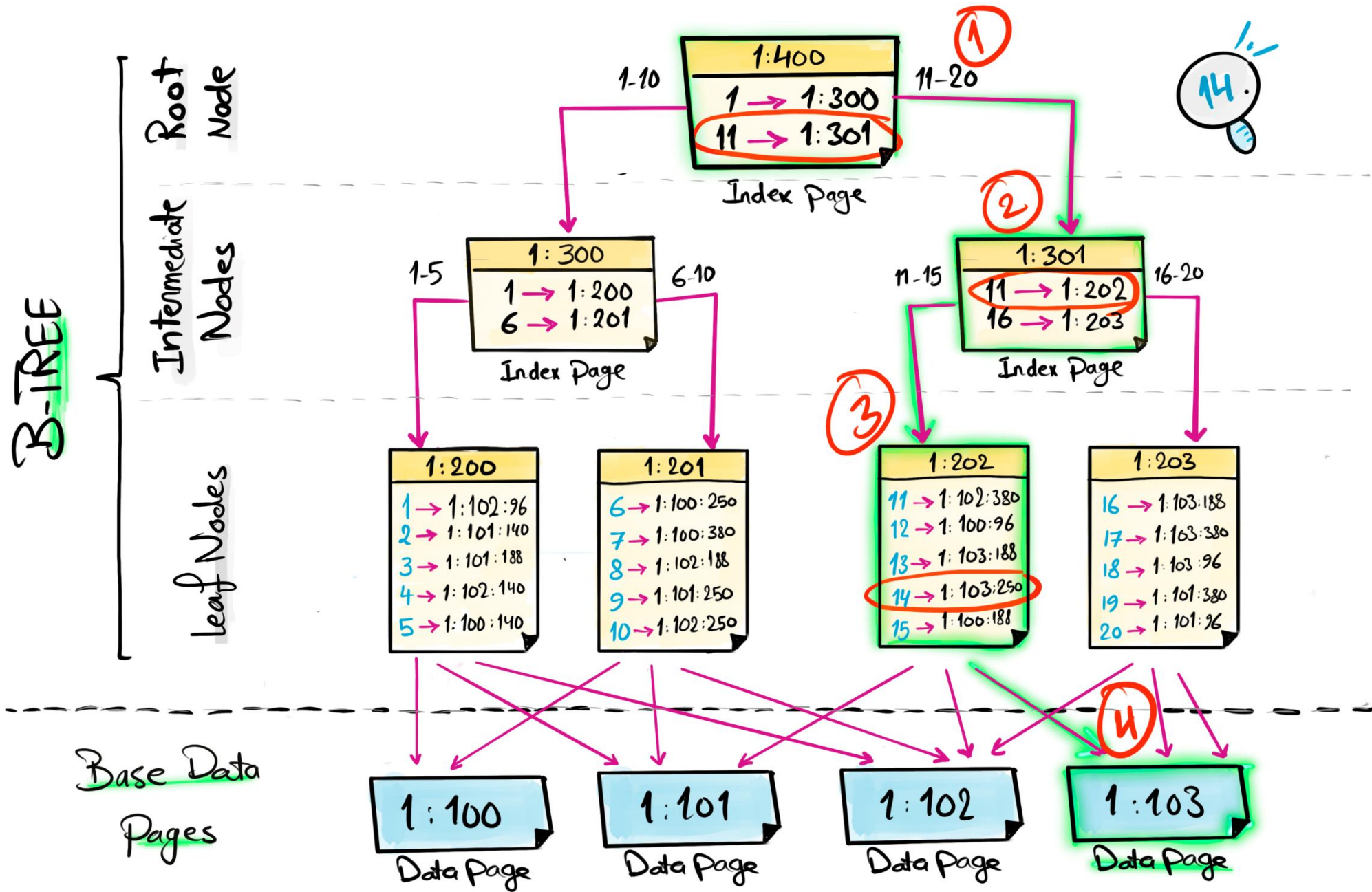


Heap structure

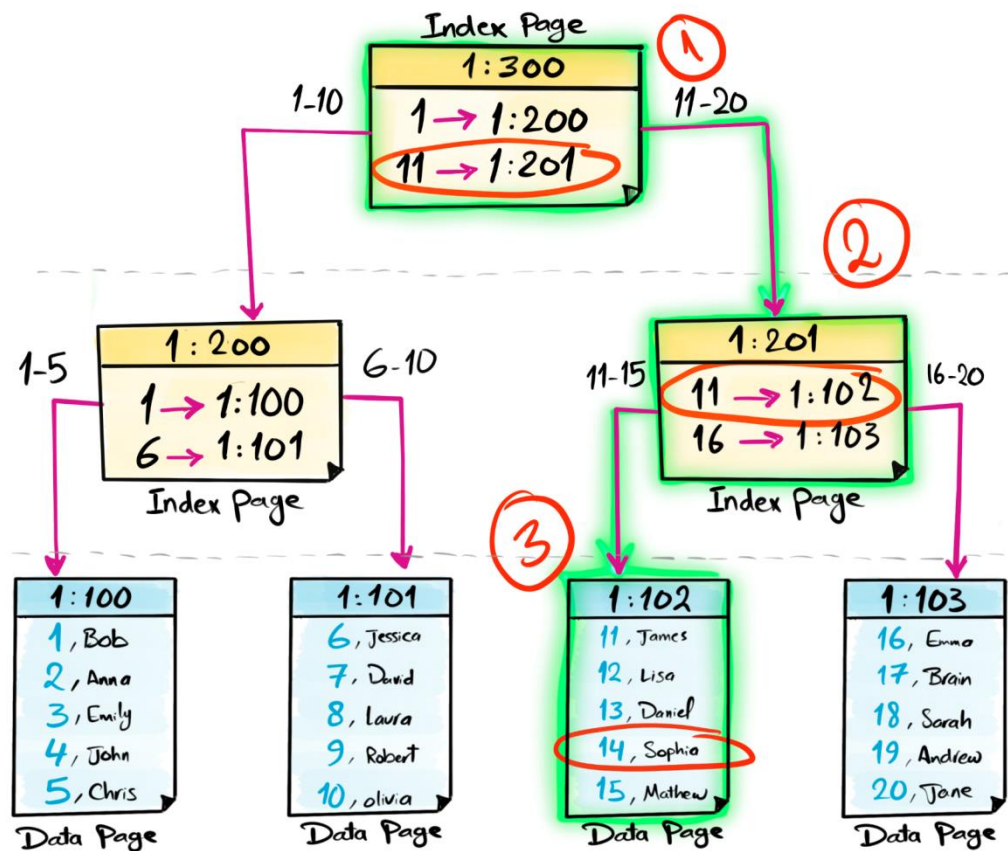




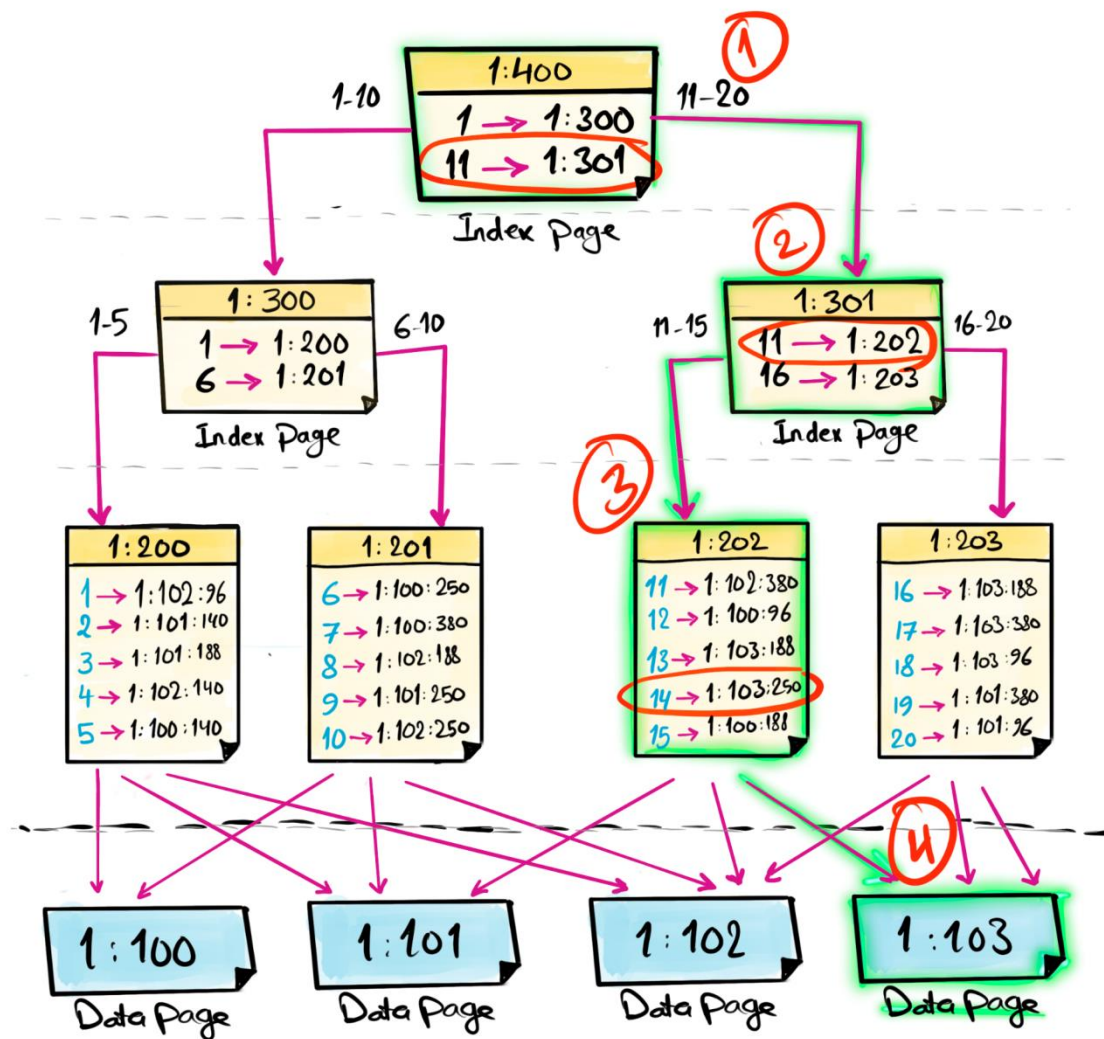




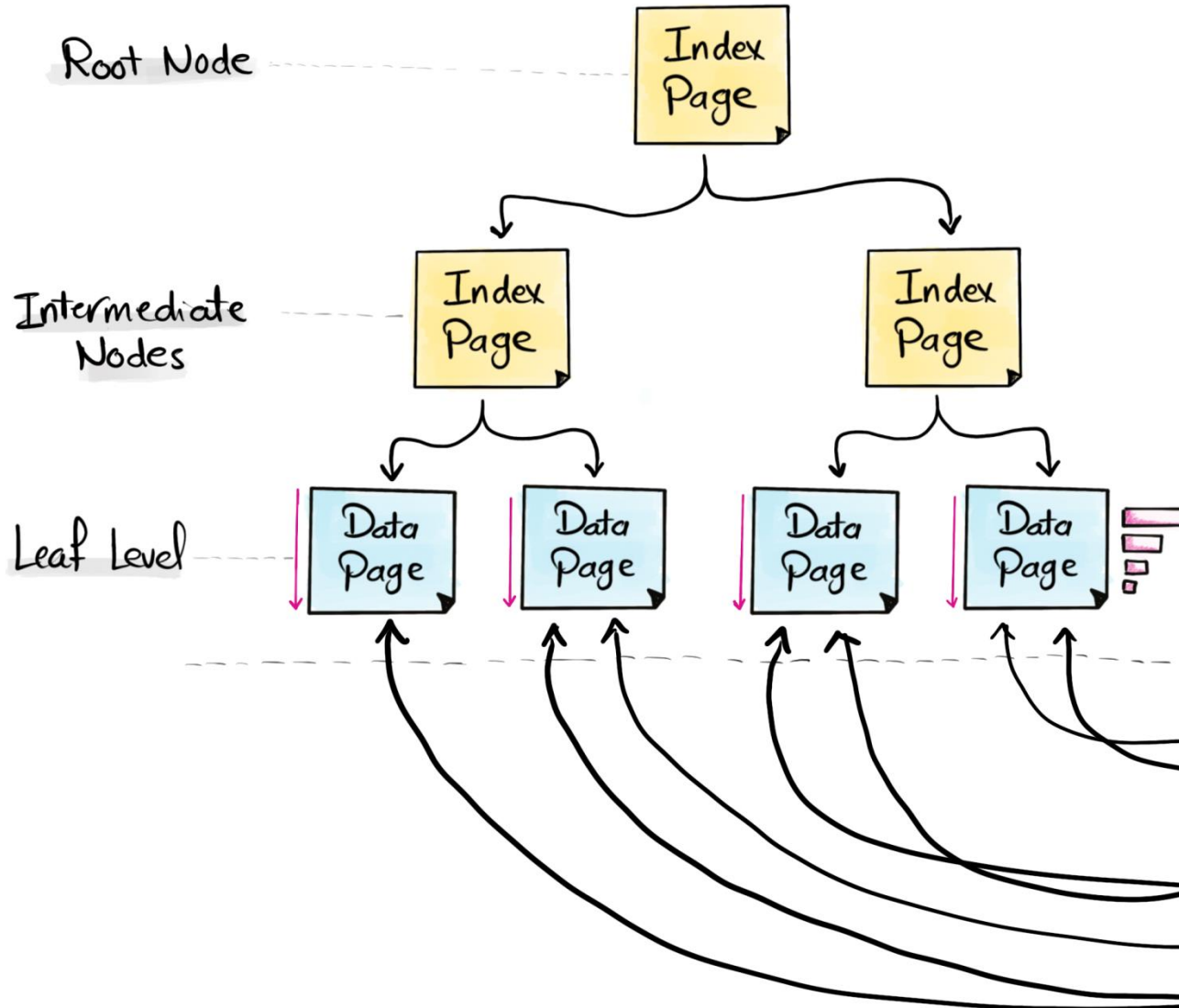
Clustered INDEX



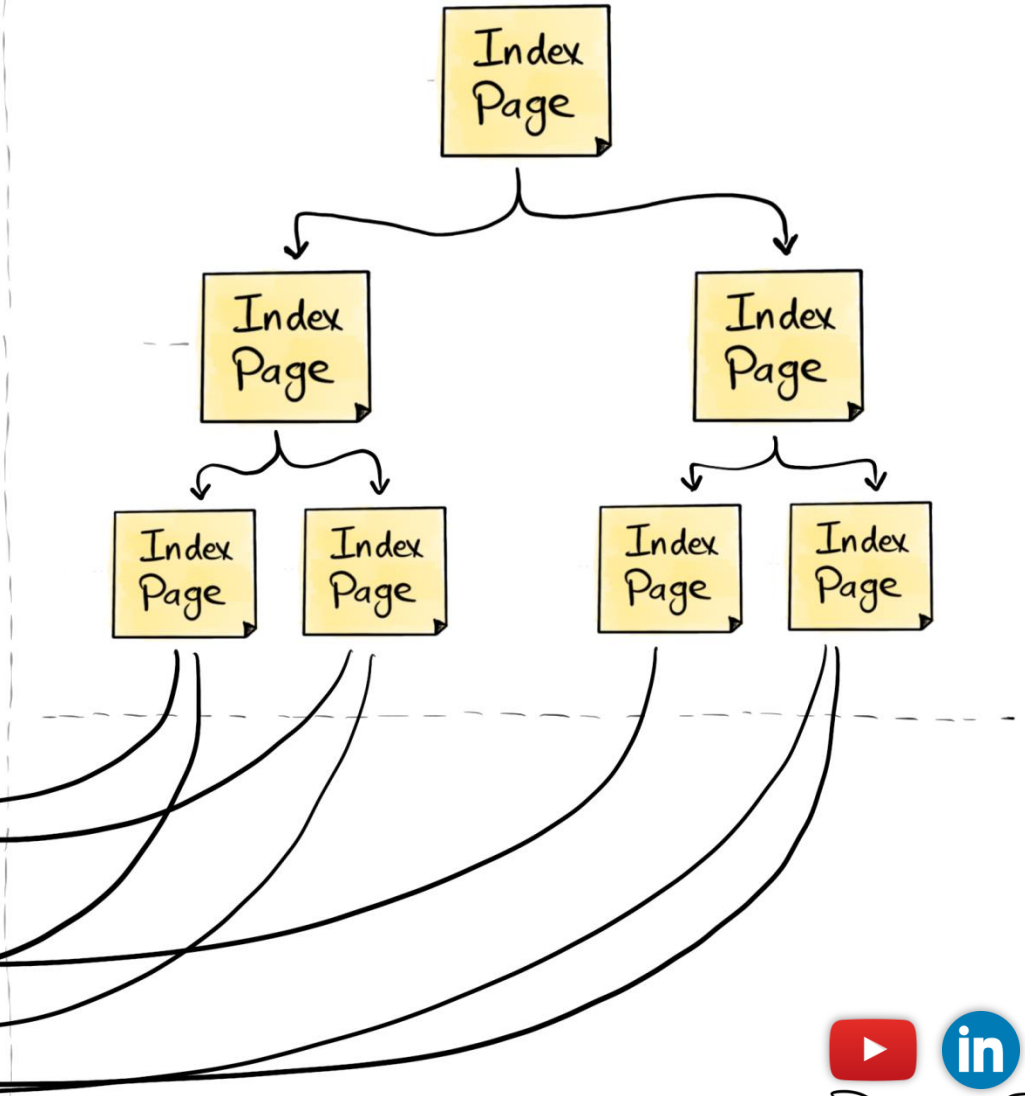
Non-Clustered Index



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	<ul style="list-style-type: none"> Unique Column Not frequently modified Column Improve range query performance 	<ul style="list-style-type: none"> 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)
```

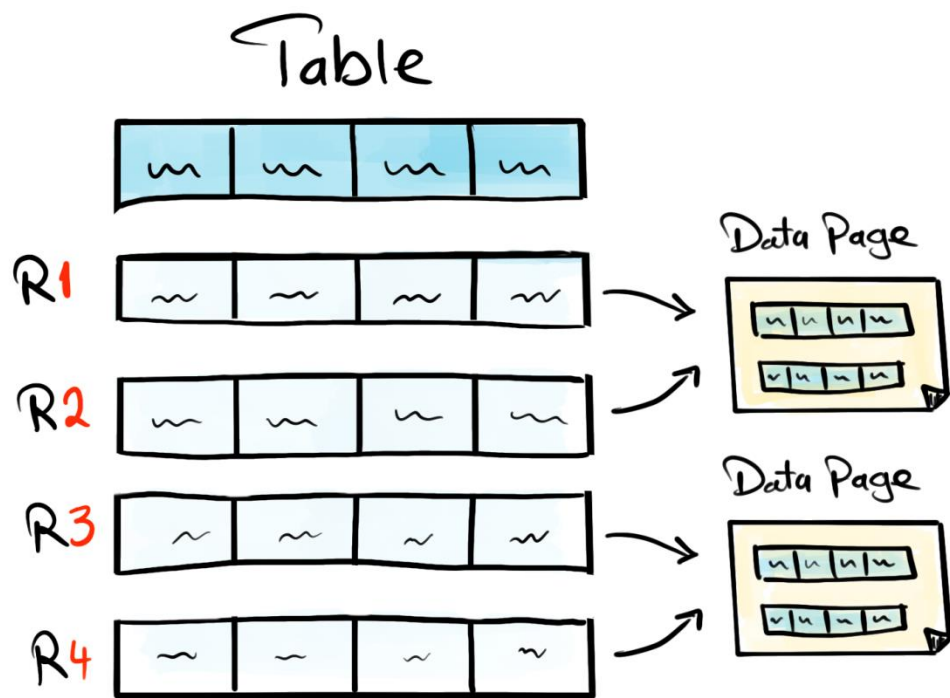
```
CREATE NONCLUSTERED INDEX IX_Customers_City ON Customers (City)
```

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

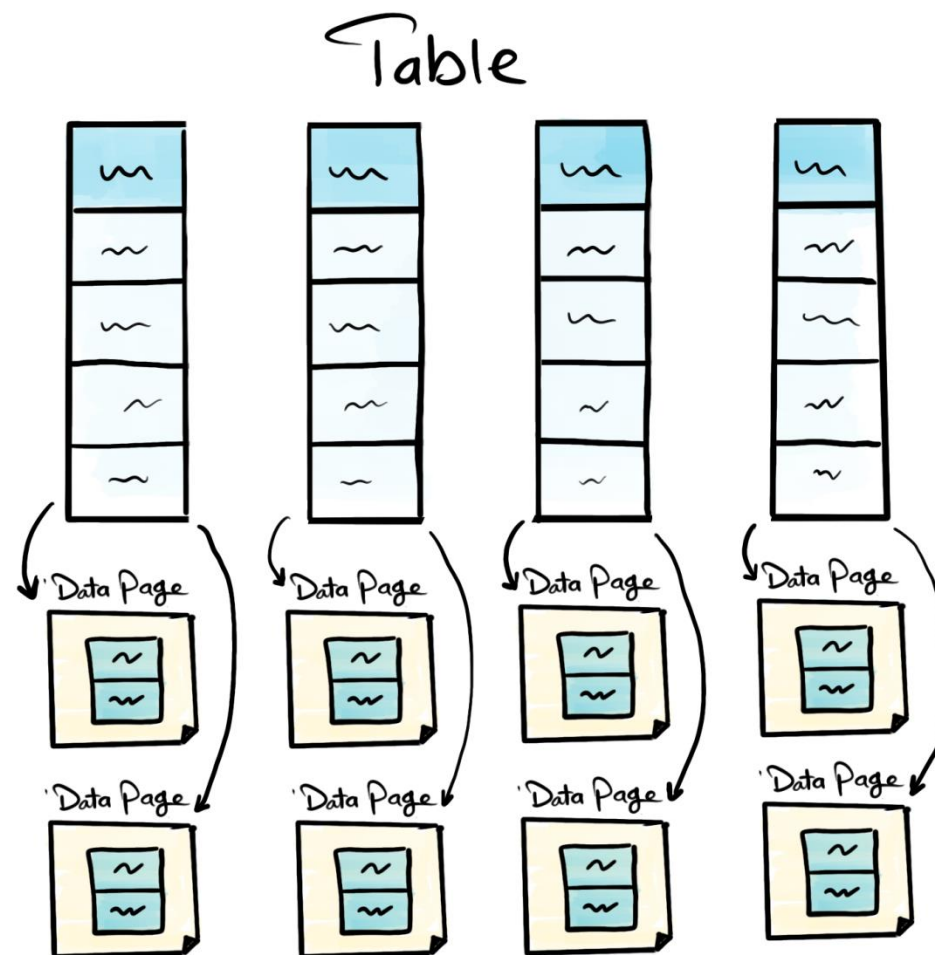


NONCLUSTERED

Rowstore Index

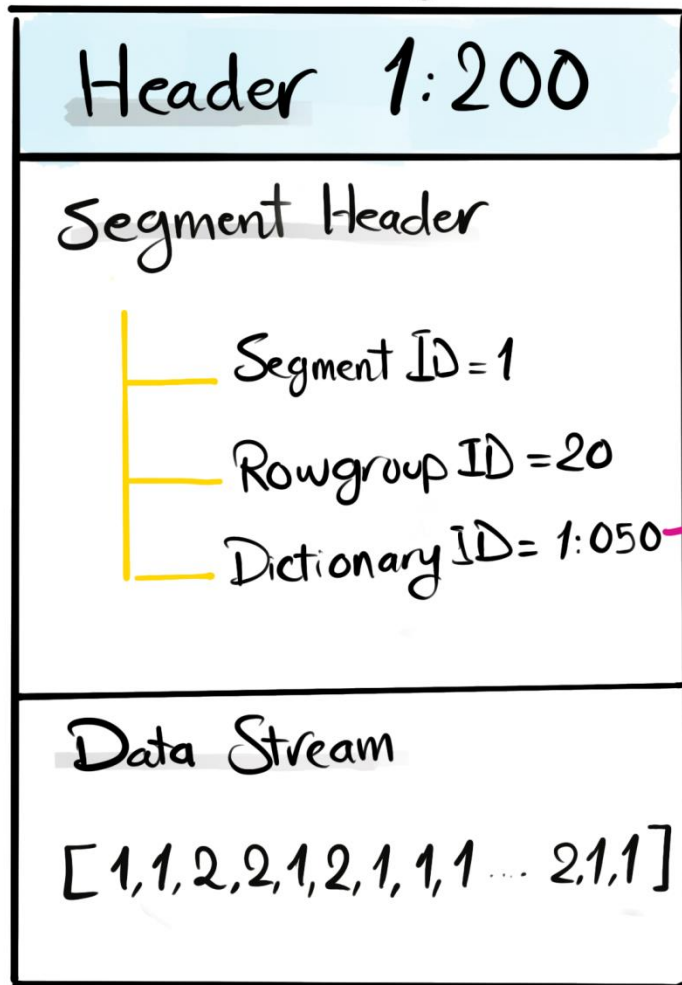


Columnstore Index

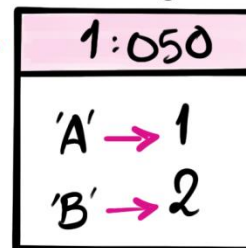


Columnstore

LoB Page

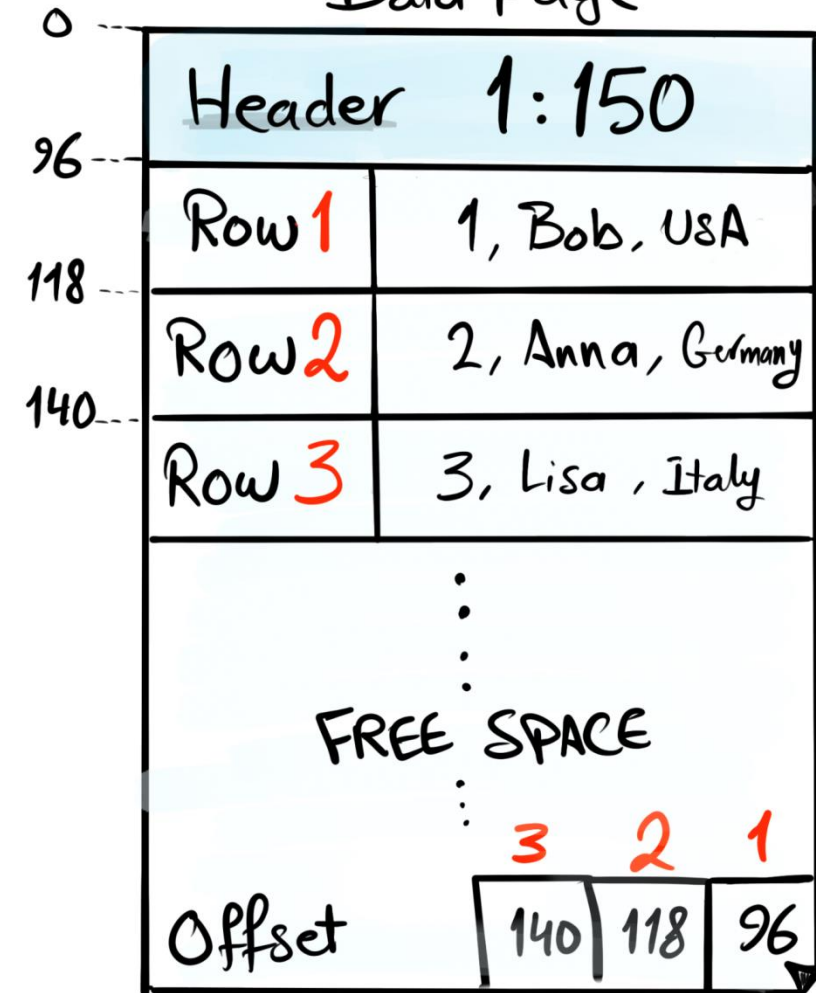


Dictionary Page



Rowstore

Data Page



Rowstore Heap Table

Diagram illustrating a 2D array structure with 2M rows and 3 columns. The array is divided into two 1M row blocks. The first block has a blue header row with columns 'ID', 'Name', and 'Status'. The second block has a red header row with columns 'ID', 'Name', and 'Status'. A red dashed line separates the two blocks.

[illegible]

ID	Name	Status
1	John	Active
2	Jane	Active
3	Bob	Active
4	Alice	Active
5	Charlie	Active
6	Diana	Active
7	Eve	Active
8	Frank	Active
9	Grace	Active
10	Henry	Active

The diagram illustrates a simple database structure with three tables: ID, Name, and Status. Each table has a header row and five data rows, representing a basic schema for storing information.

The diagram illustrates three separate data structures, each represented as a vertical table with a header and six data rows. The first table is labeled 'ID', the second 'Name', and the third 'Status'. Each data row contains a wavy line, representing a value that is not yet defined or is a placeholder.

The diagram shows three vertical columns representing database fields. The first column is labeled 'ID' and contains 8 rows of data, each represented by a wavy line. The second column is labeled 'Name' and contains 5 rows of data, each represented by a wavy line. The third column is labeled 'Status' and contains 2 rows of data, each represented by a wavy line.

The diagram illustrates three separate database tables. The first table, labeled 'ID', contains six rows of data, each represented by a wavy line. The second table, labeled 'Name', contains three rows of data, each represented by a wavy line. The third table, labeled 'Status', contains two rows of data, each represented by a wavy line.

LoB Page 1:200

LoB Page 1:201

LoB Page 1:202

LoB Page 1:200

LoB Page 1:201

LoB Page 1:202

ID: [1,2,3,4, 5,6,7,8]

Name: [1,2,3,4, 5,5,9,6]

Status: [1,1,2,1, 2,1,1,1]

TABLE

ID	Name	Status
1	John	Active
2	Sarah	Inactive
3	Alex	Active
4	Emma	Active
5	James	Inactive

Columnstore

Data Page 1: 200

Dictionary
Data [1, 2, 3, 4, 5]

Data Page 1: 201

Dictionary 1 = 'John' 2 = 'Sarah' 3 = 'Alex' 4 = 'Emma' 5 = 'James'
Data [1, 2, 3, 4, 5]

Data Page 1: 202

Dictionary 1 = 'Active' 2 = 'Inactive'
Data [1, 2, 1, 1, 2]

[1, ~~X~~, 1, 1, ~~X~~] ⇒ 3

Rowstore

Data Page 1: 100

[1, John, Active]
[2, Sarah, Inactive]

Data Page 1: 101

[3, Alex, Active]
[4, Emma, Active]

Data Page 1: 102

[5, James, Inactive]

[1, John, Active]
~~[2, Sarah, Inactive]~~
 [3, Alex, Active]
 [4, Emma, Active]
~~[5, James, Inactive]~~
 ⇒ 3

QUERY

SELECT COUNT (*) FROM Customers
 WHERE Status = 'Active'

Rowstore Index

Columnstore Index

Definition

Organizes and stores data
row by row

Organizes and stores data
column by column

Storage Efficiency

Less efficient in storage

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

Default is
ROWSTORE

Unique Index

CRE

Rowstore

Ensures no duplicate values exist in specific column. (country)

Benefits

Columnstore

1. Enforce uniqueness
2. Slightly increase query performance

customers (Country)

NOT ALLOWED
TO USE
COLUMNS

Rules

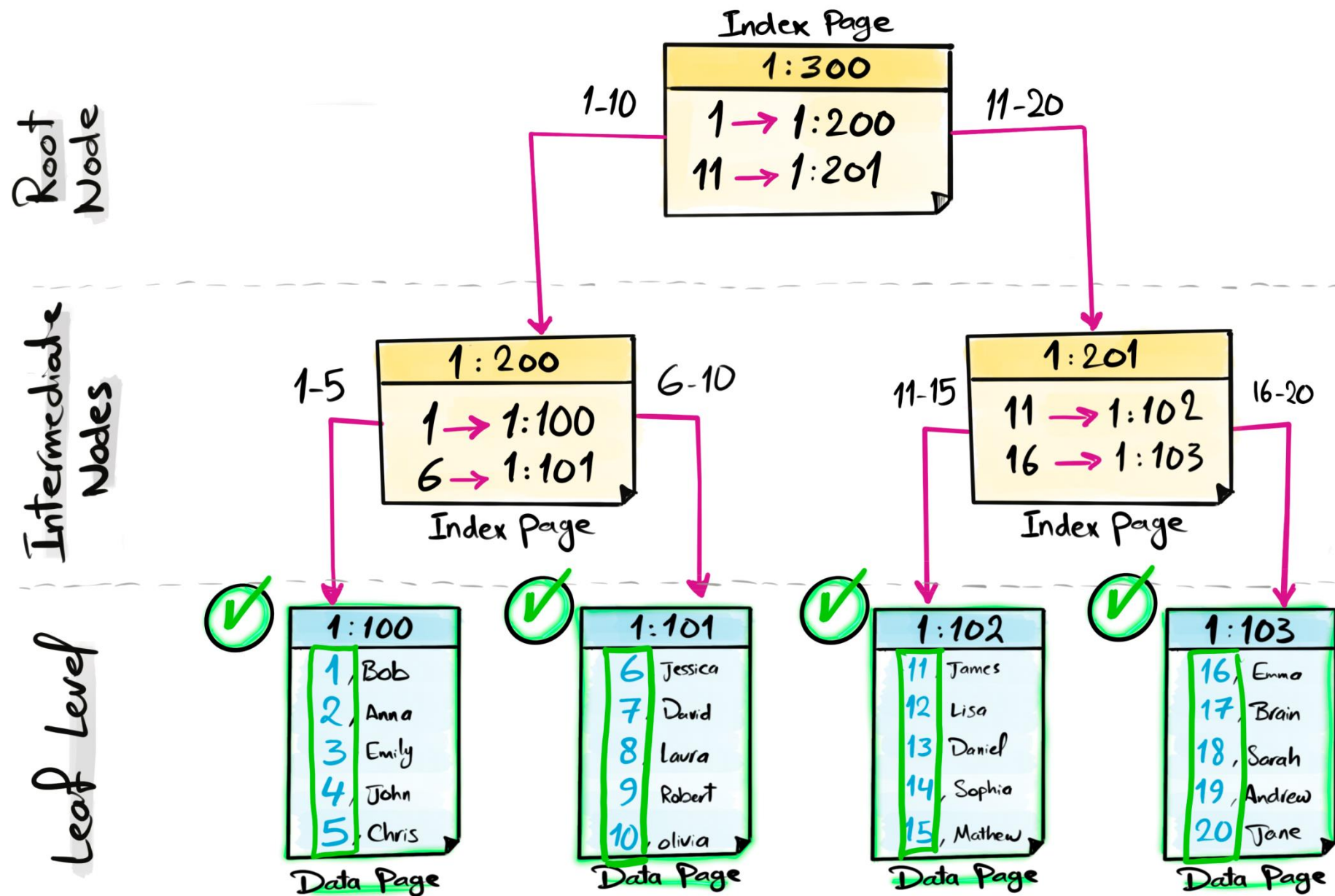
- You **can't** specify columns in Clustered Index Columnstore

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**.

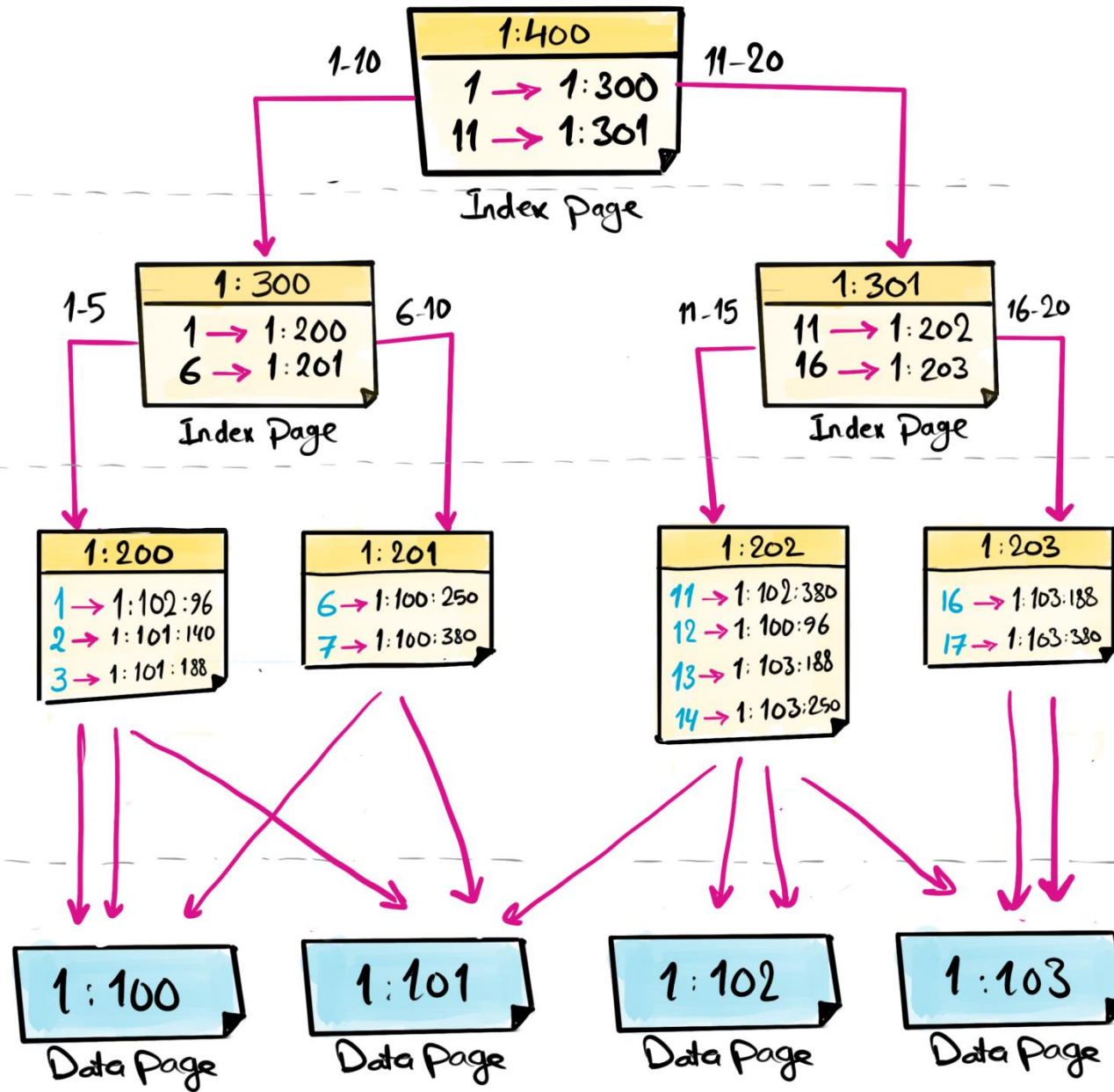
B-TREE

Root Node

Intermediate Nodes

Leaf Nodes

Base Data Pages



CONDITION

Status = 'Active'



When To Use

HEAP

Fast **Inserts**
(For Staging Tables)

MAIN

Clustered Index

OLTP

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

Columnstore Index

OLAP

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

Non-Clustered Index

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

Filtered Index

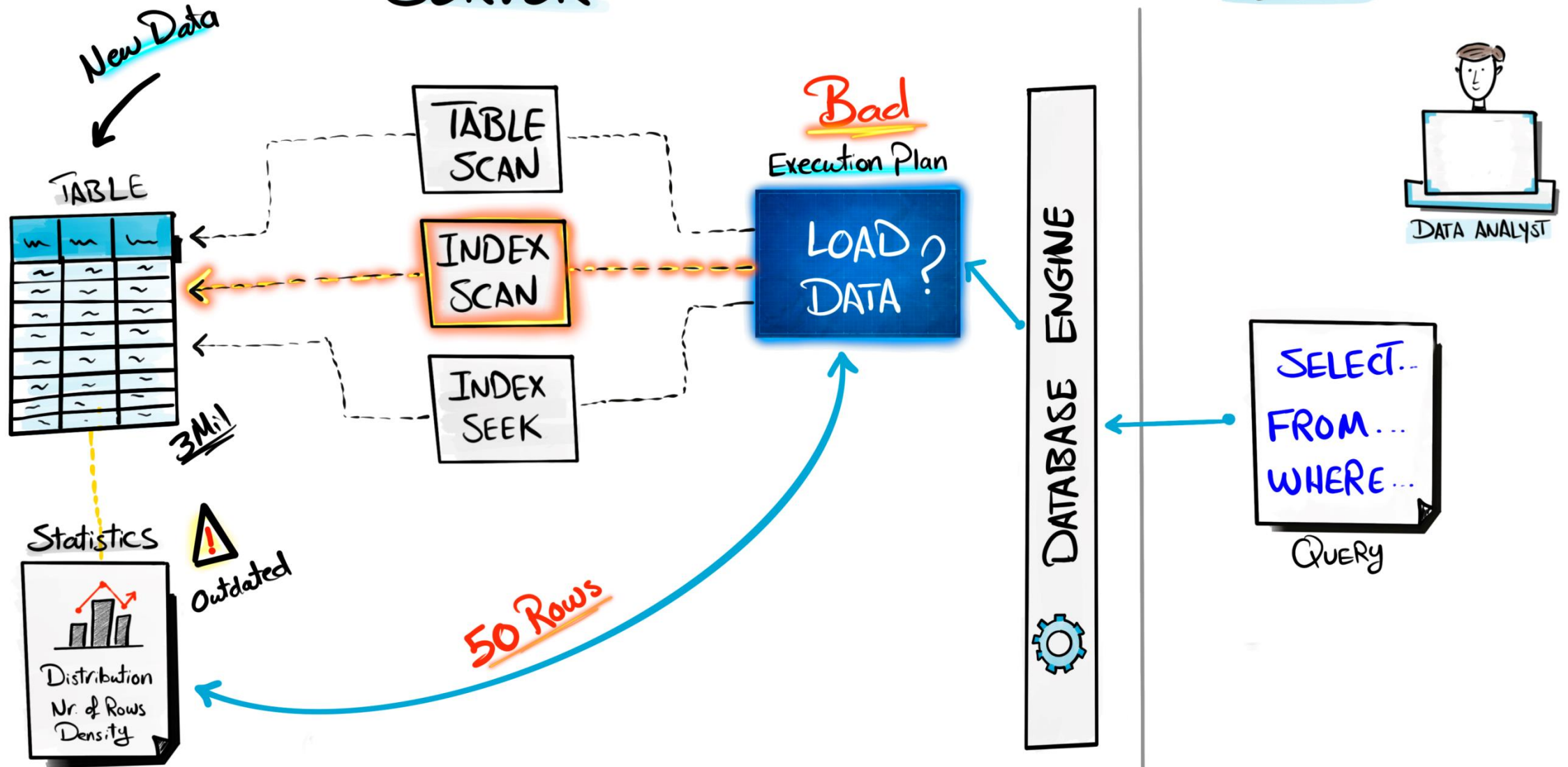
Target **Subset** of Data
Reduce Size of Index

Unique Index

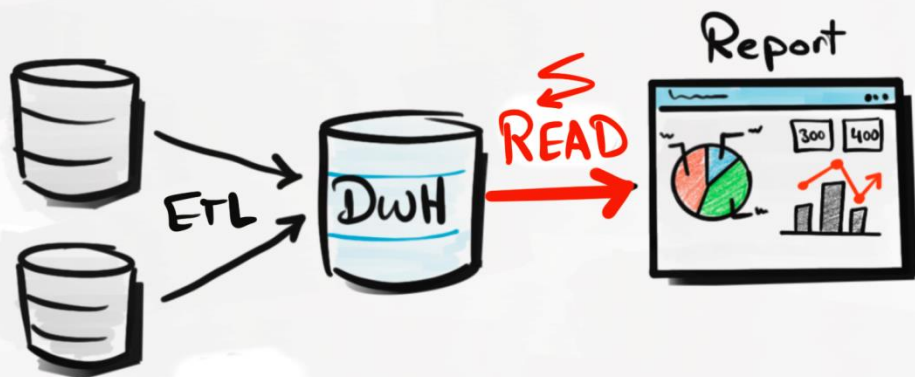
Enforce **Uniqueness**
Improve Query Speed

SERVER

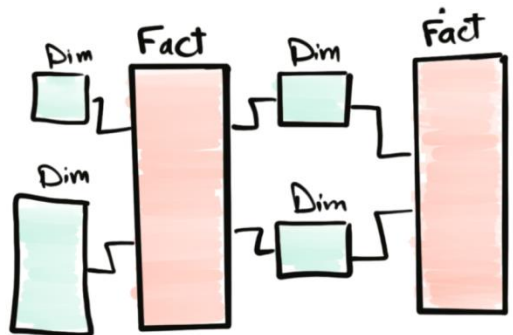
CLIENT



OLAP (Analytical)



COLUMNSTORE INDEX



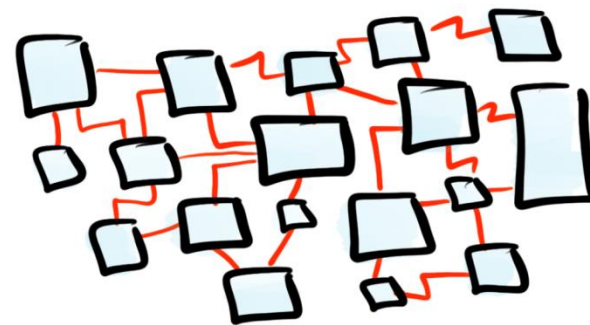
GOAL

Optimize **READ** Performance

OLTP (Transaction)



CLUSTERED INDEX PK



GOAL

Optimize **WRITE** Performance

#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

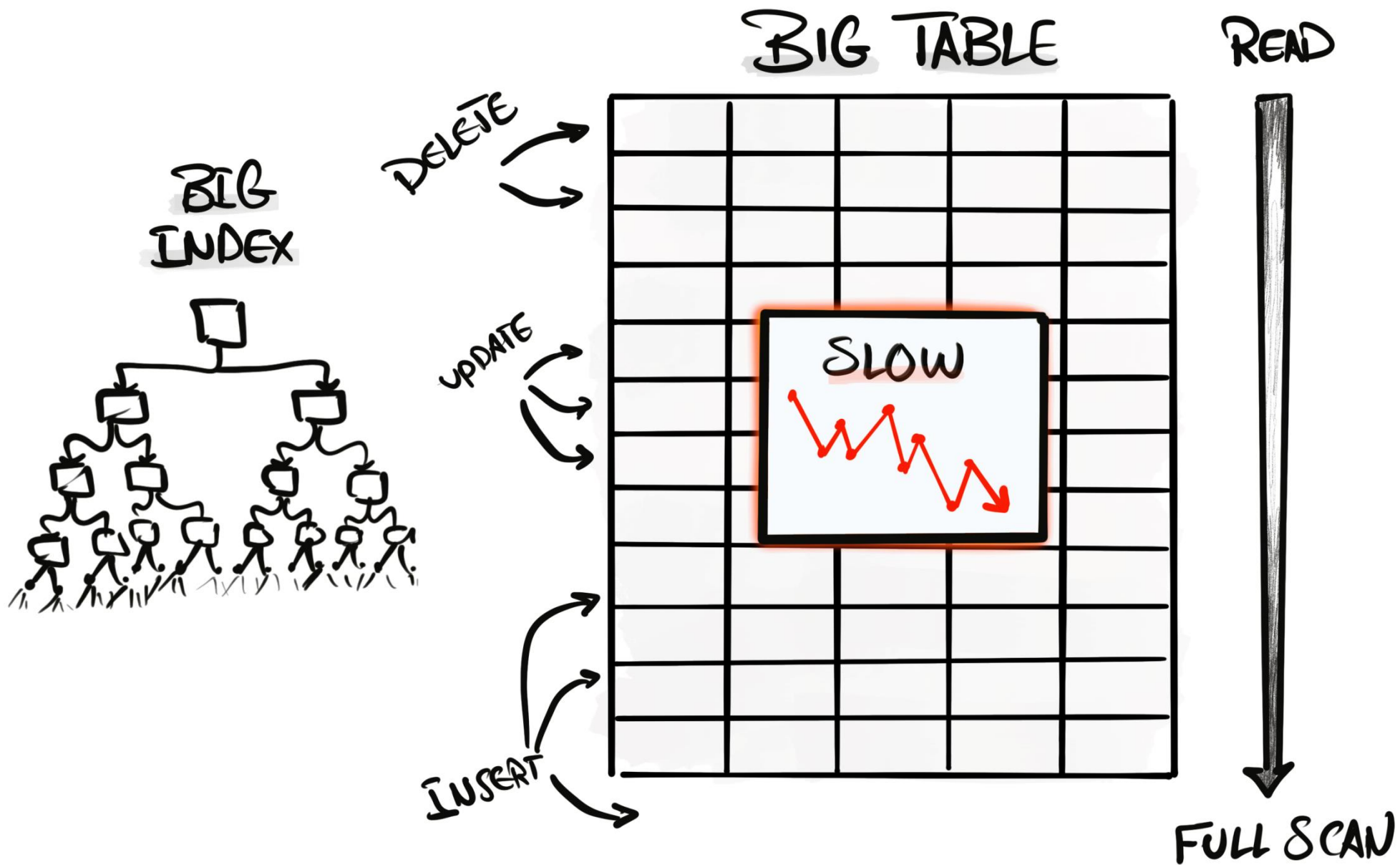
Partitioning

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

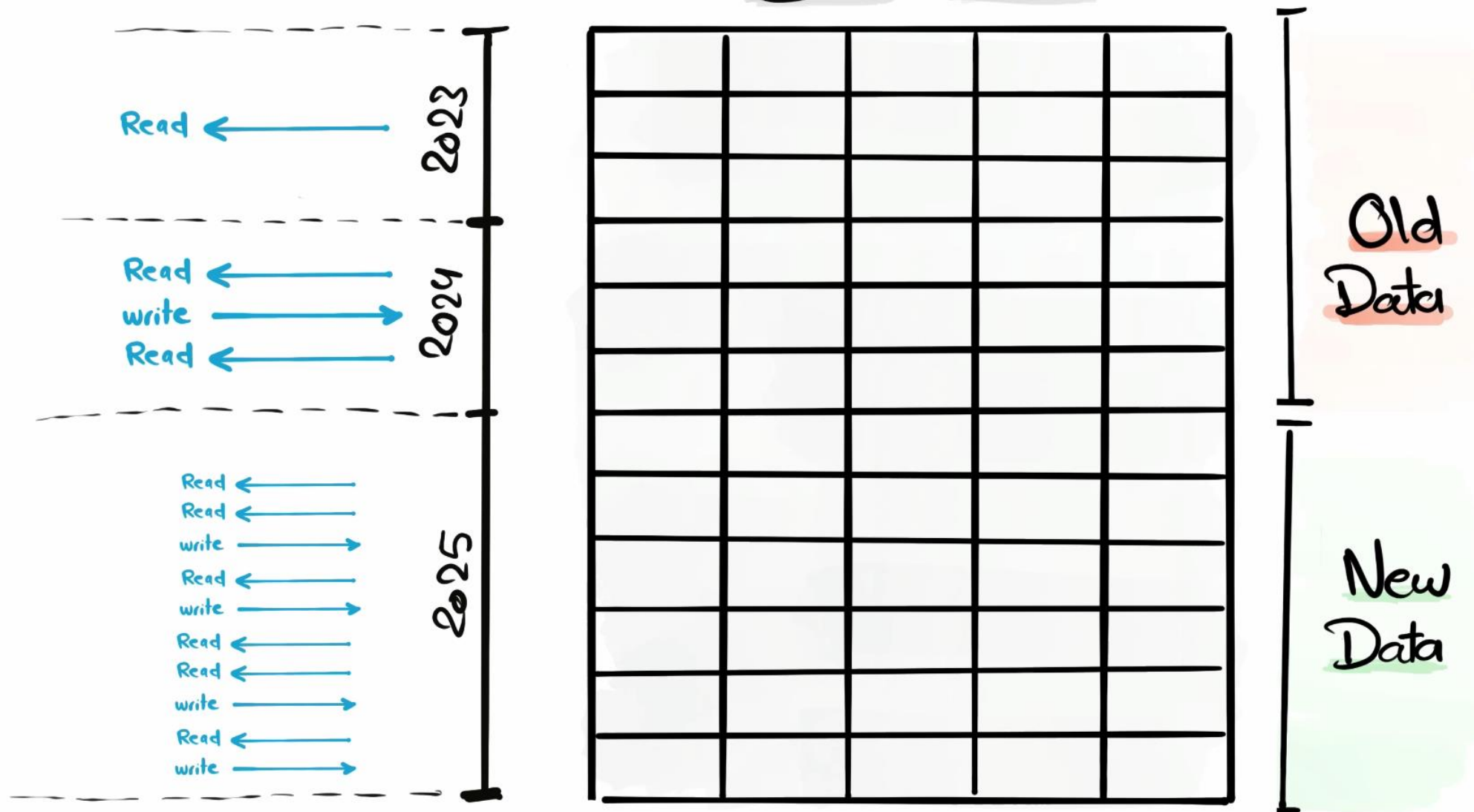


SQL PARTITIONING

Divides Big Table into Smaller Partitions
while still being treated as a single logical table.

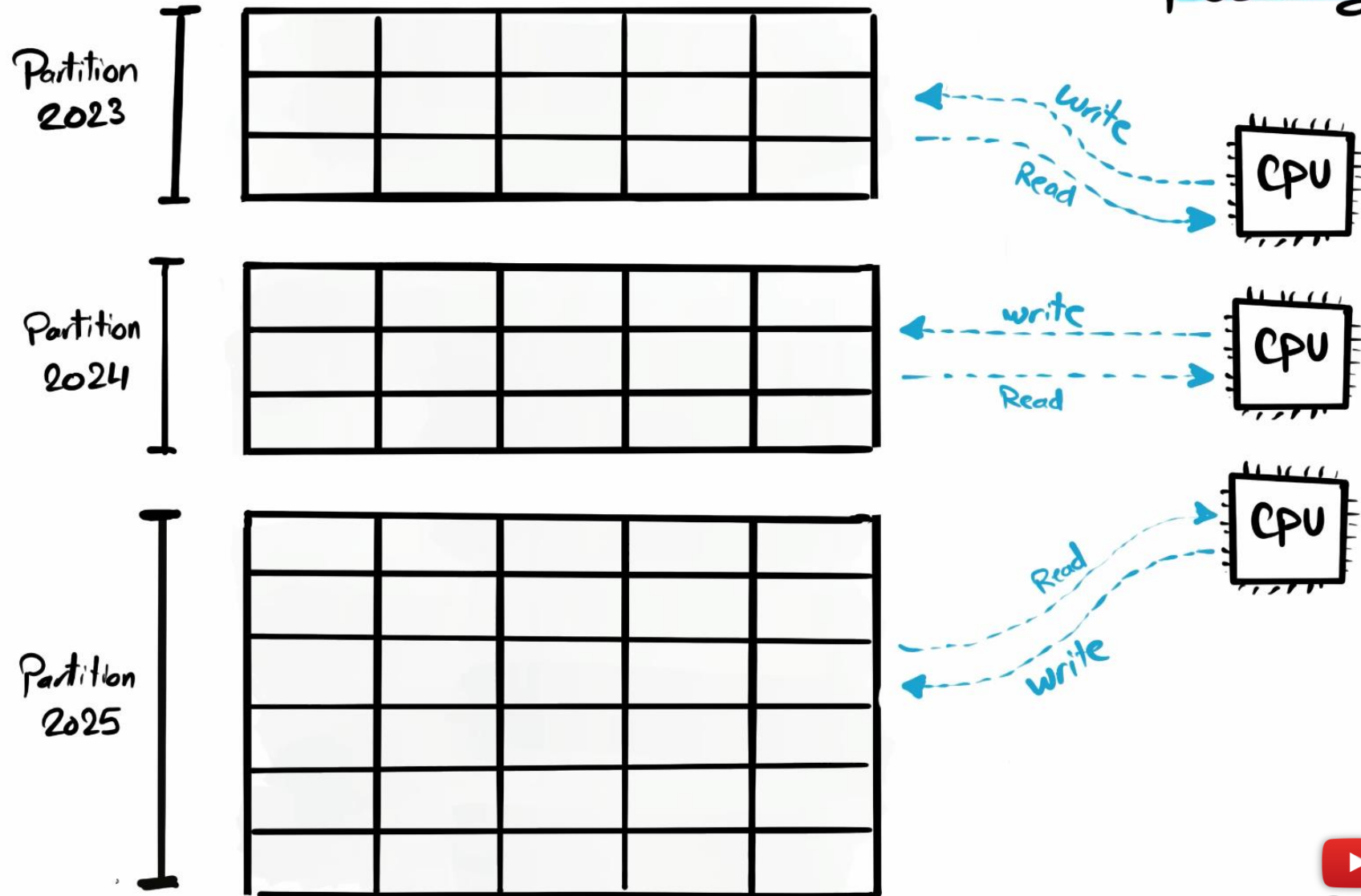


BIG TABLE



BIG TABLE

Parallel
processing



BIG TABLE

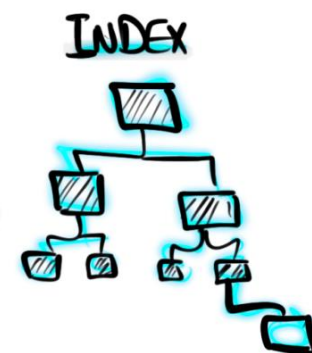
Partition
2023



Partition
2024



Partition
2025



NEW
DATA



PARTITION FUNCTION

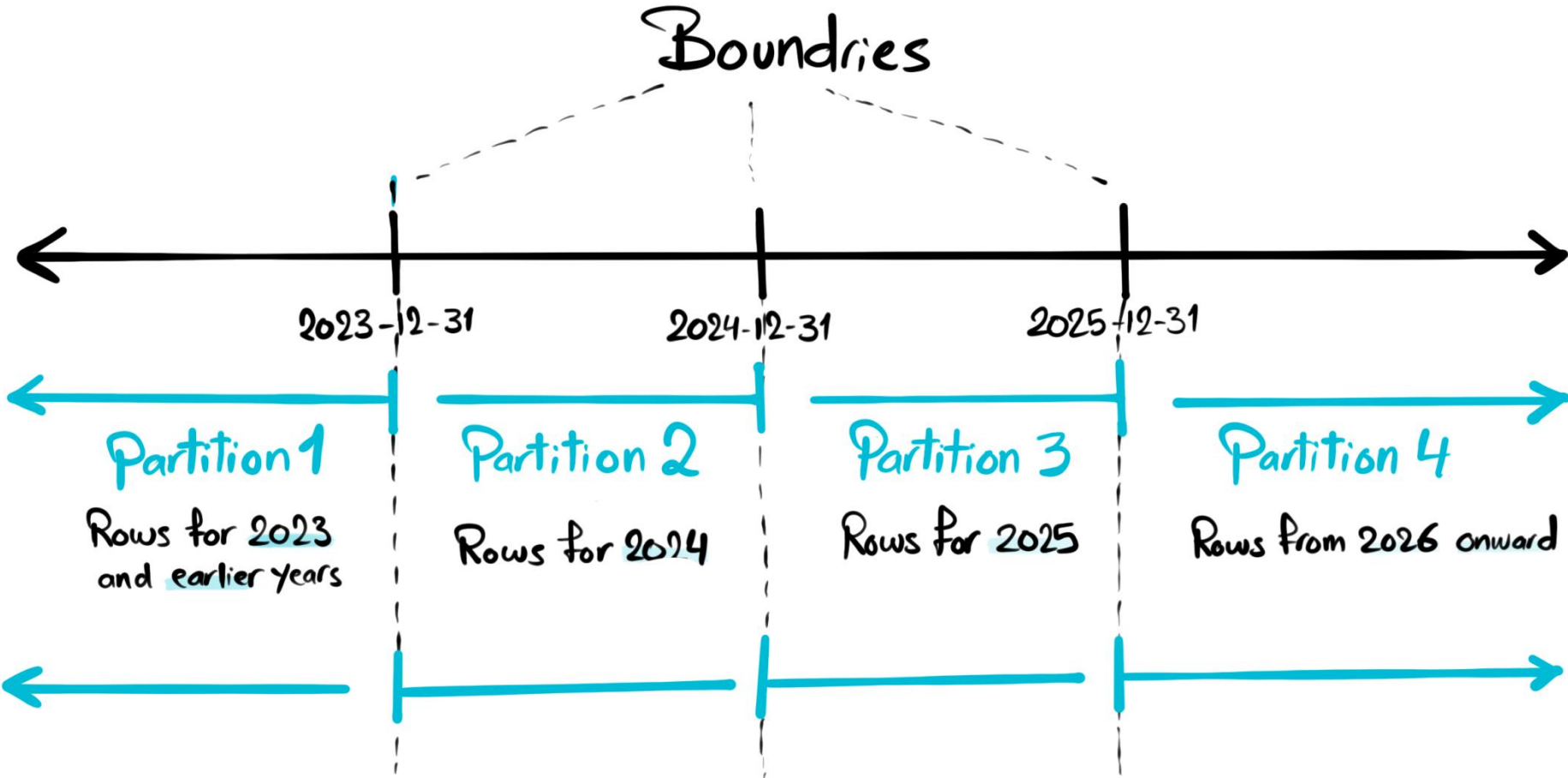
Define the **Logic** on how to divide
your data into partitions !

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

① Partition Function

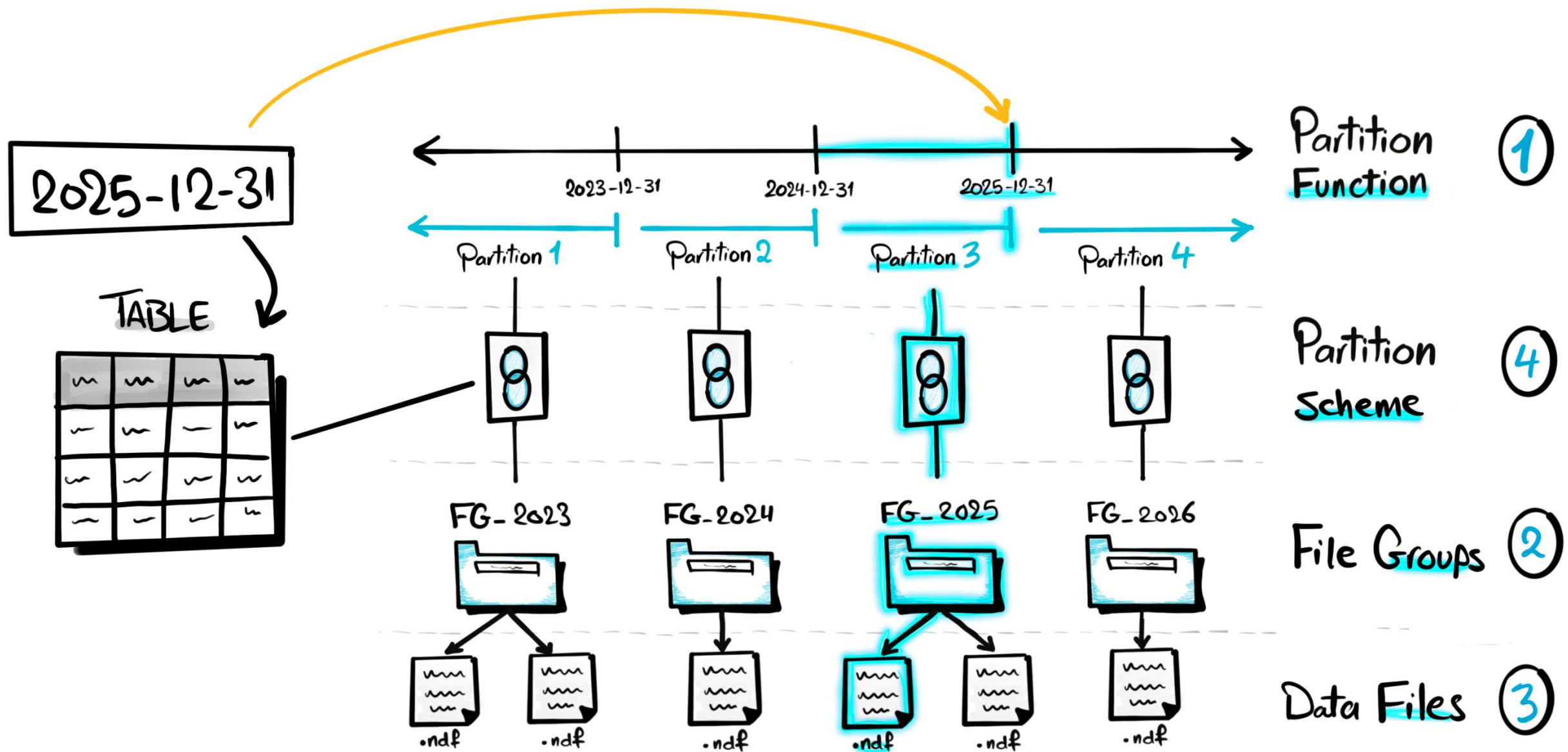
LEFT Logic

RIGHT Logic

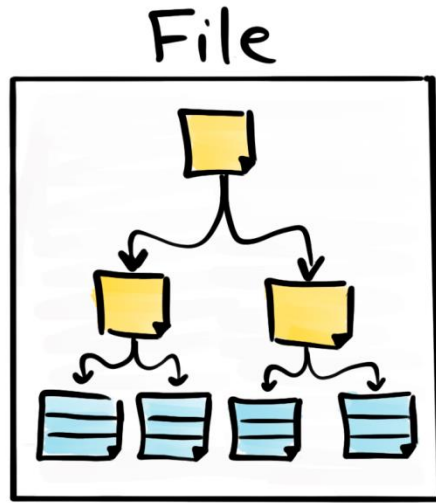


FILEGROUPS

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



Indexing



Rowstore Index

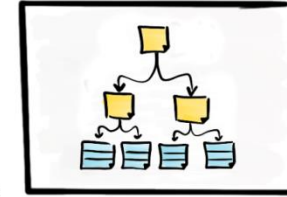
TABLE

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

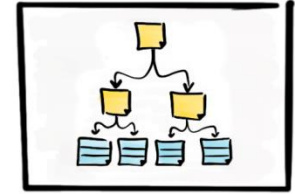
Horizontal Partitioning

Partitioning + Indexing

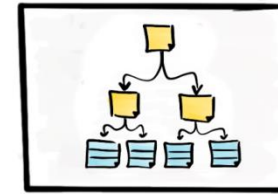
File1



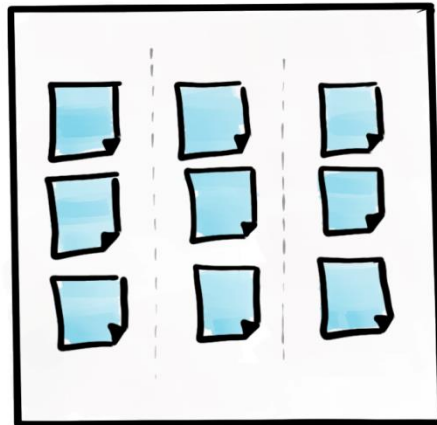
File2



File3



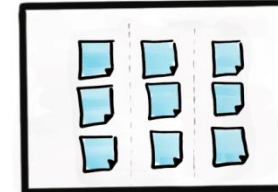
File



Columnstore Index

Vertical Partitioning

File1



File2

