Indexing

Indexing

- A SQL index is a quick lookup table for finding records users need to search frequently.
- An index is small, fast, and optimized for quick lookups.
- It is very useful for connecting the relational tables and searching large tables.
- Simply put, an index is a pointer to data in a table.
- An index helps to speed up **SELECT** queries and **WHERE** clauses, but it slows down data input, with the **UPDATE** and the **INSERT** statements.
- **Indexes can be created or dropped with no effect on the data.**
- SQL Server supports several types of indexes but one of the most common types is the clustered index.
- This type of index is automatically created with a primary key.

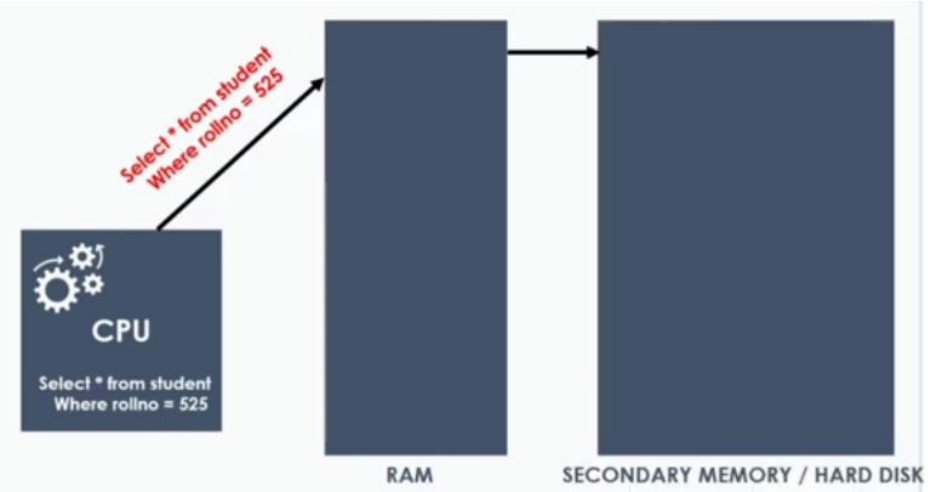
Why we need Indexing in DBMS?

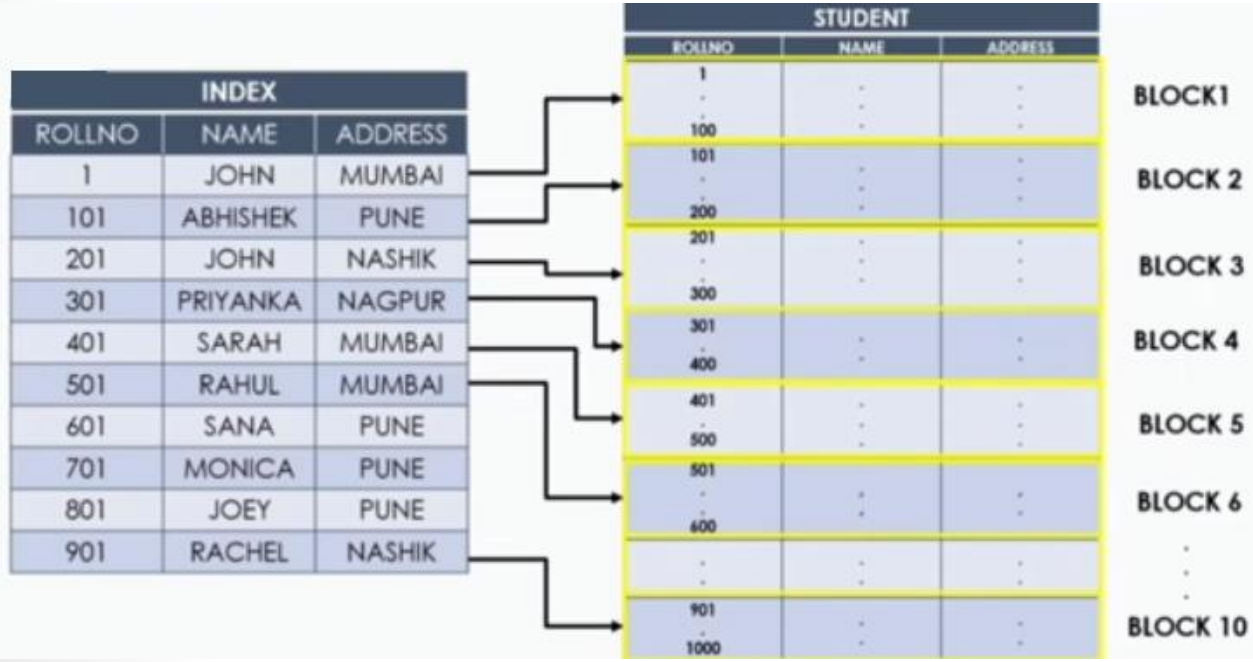
- Suppose, we want to search record of student whose roll no. is 525 from 1000 records in a table.

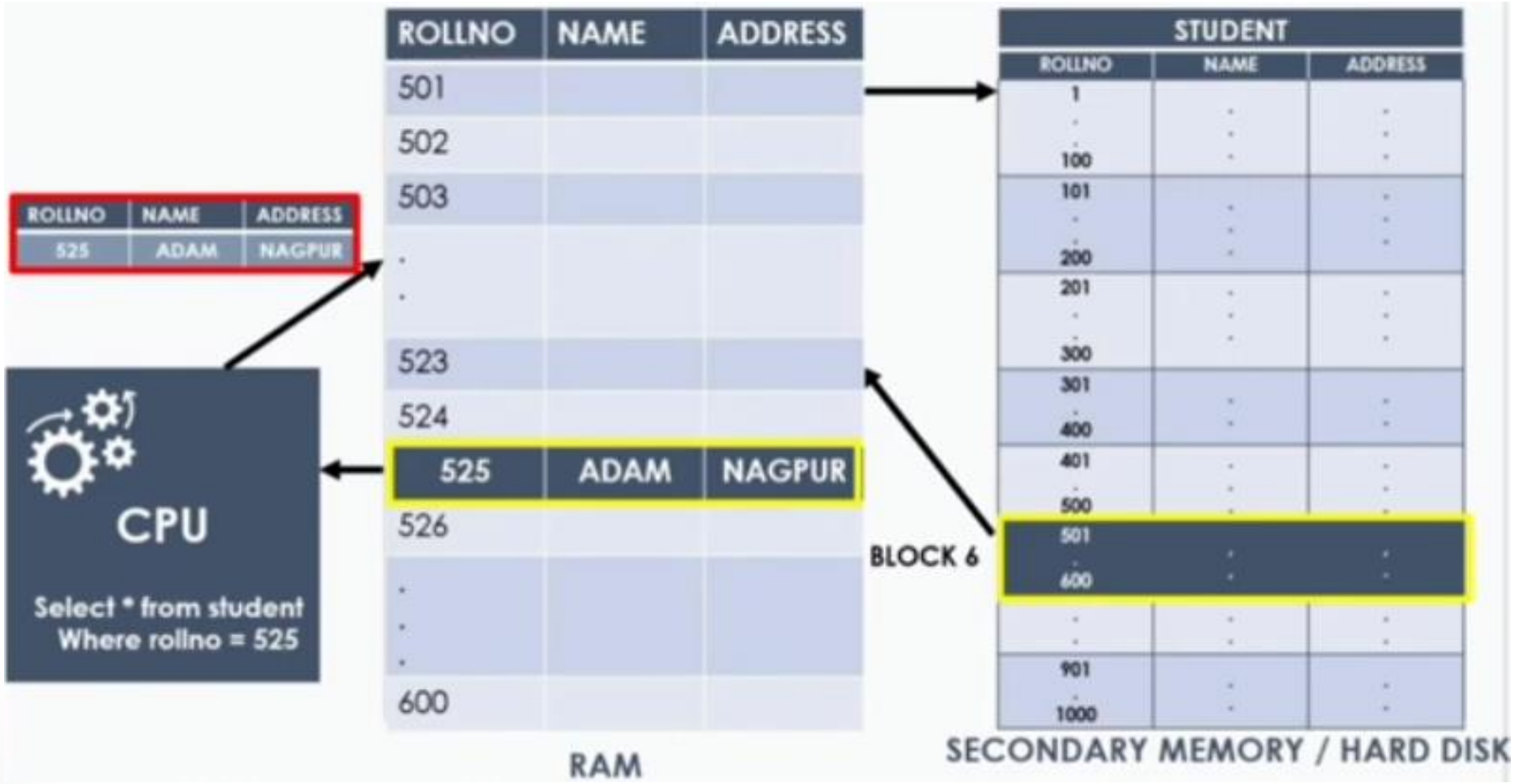
STUDENT		
ROLLNO	NAME	ADDRESS
1	JOHN	MUMBAI
2	SMITH	PUNE
3	DAISY	NASHIK
.	.	.
.	.	.
.	.	.
523	MONICA	PUNE
524	ROHN	MUMBAI
525	ADAM	NAGPUR
.	.	.
.	.	.
.	.	.
1000	JOHNSON	MUMBAI

- Query: select * from student where rollno=525;
- It will search 524 records in a table then we get 525^{th} record.
- Time consuming

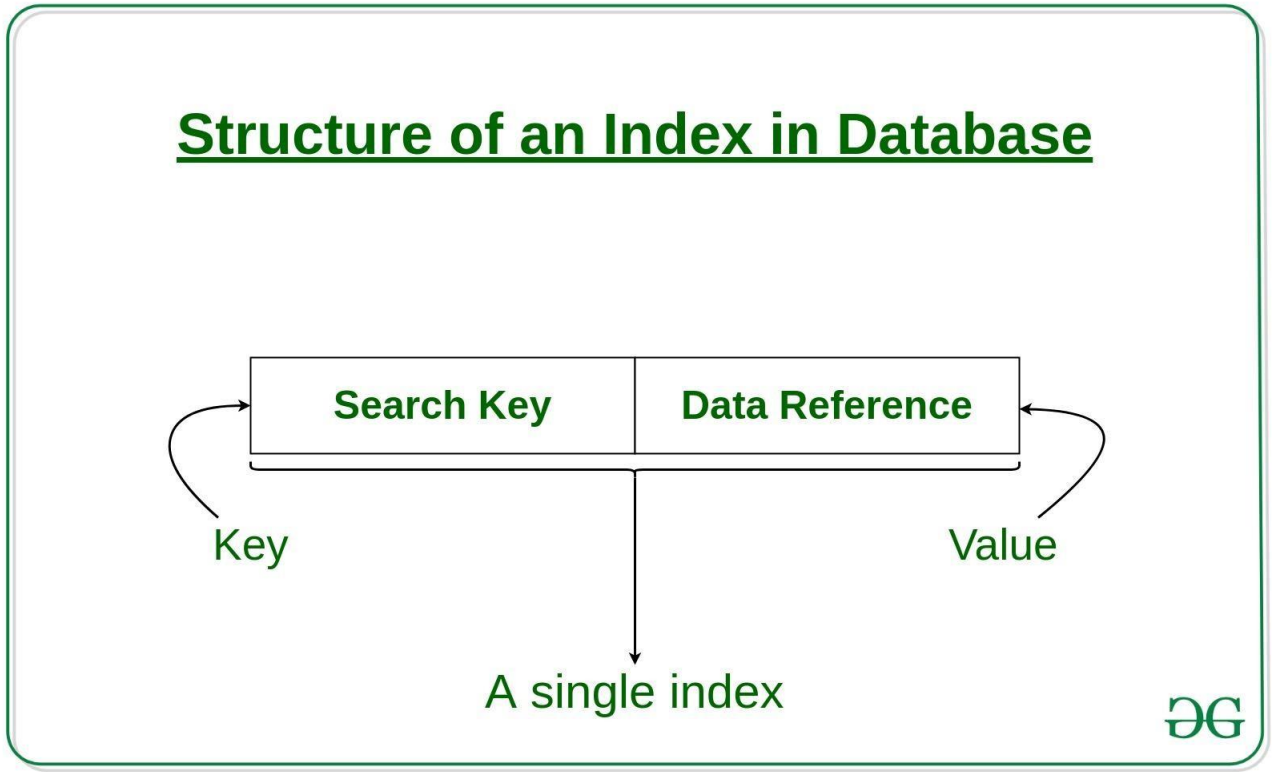
Solution: Indexing






ROLLNO
NAME
ADDRESS
525
ADAM
NAGPUR
CPU
Select * from student
Where rollno = 525
ROLLNO
NAME
ADDRESS
501
502
503
523
524
525
ADAM
NAGPUR
526
600
RAM
BLOCK 6
STUDENT
ROLLNO
NAME
ADDRESS
1
100
101
200
201
300
301
400
401
500
501
600
901
1000
SECONDARY MEMORY / HARD DISK

- Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- It is a data structure technique which is used to quickly locate and access the data in a database.



The indexing has various attributes:

- **Access Types**: This refers to the type of access such as value based search, range access, etc.
- **Access Time**: It refers to the time needed to find particular data element or set of elements.
- **Insertion Time**: It refers to the time taken to find the appropriate space and insert a new data.
- **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead**: It refers to the additional space required by the index.

Ordered indices

- Ordered index similar to index in a textbook
- An index access structure is usually defined on a single field of a file, called an indexing field (or indexing attribute).
- The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value.
- There are several types of indexes.
  - Primary Index
  - Clustering Index
  - Secondary Index

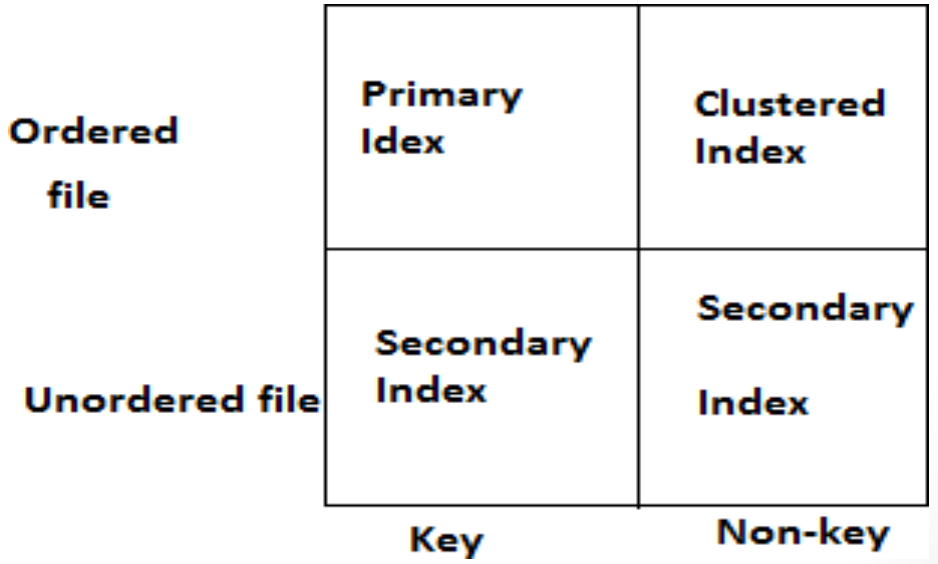
- **A primary index** is specified on the ordering key field of an ordered file of records.
- An ordering key field is used to physically order the file records on disk, and every record has a unique value for that field.

- If the ordering field is not a key field—that is, if numerous records in the file can have the same value for the ordering field— another type of index, called a clustering index, can be used.
- The data file is called **a clustered file** in this latter case. Notice that a file can have at most one physical ordering field, so it can have at most one primary index or one clustering index, but not both.

- A third type of index, called **a secondary index**, can be specified on any nonordering field of a file.
- A data file can have several secondary indexes in addition to its primary access method.

Types of indices

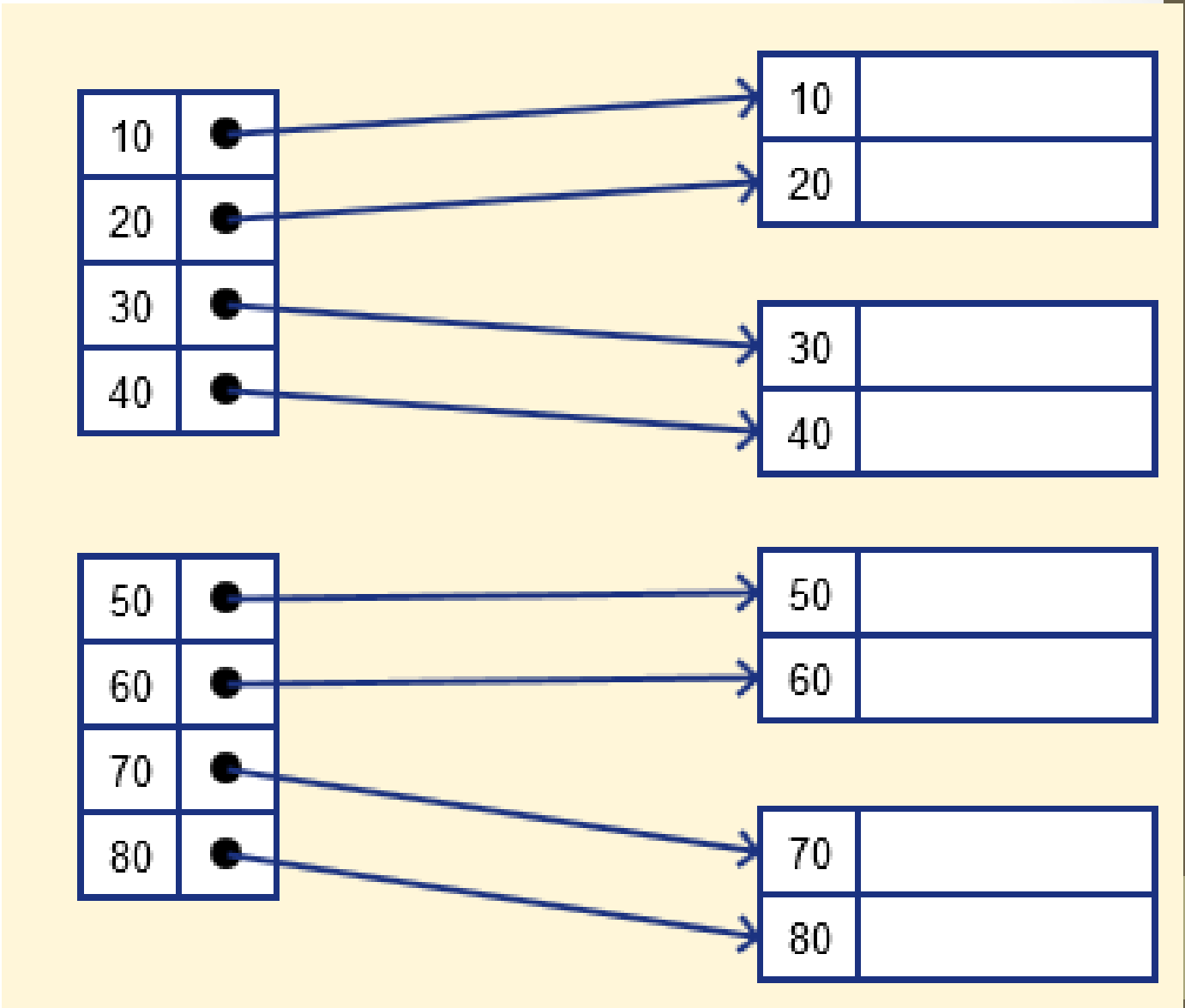
PRIMARY INDEX	CLUSTER INDEX	SECONDARY INDEX
Defined on an ordered data file, which is based on a **PRIMARY KEY** field of the relation.	Defined on an ordered data file, which is based on a **NON-KEY** field.	Defined on an un-ordered data file, generated from a **CANDIDATE KEY** of the relation



- Indexes may be dense or sparse
  - Dense index has an index entry for every search key value in the data file
  - Sparse index has entries for only some search values

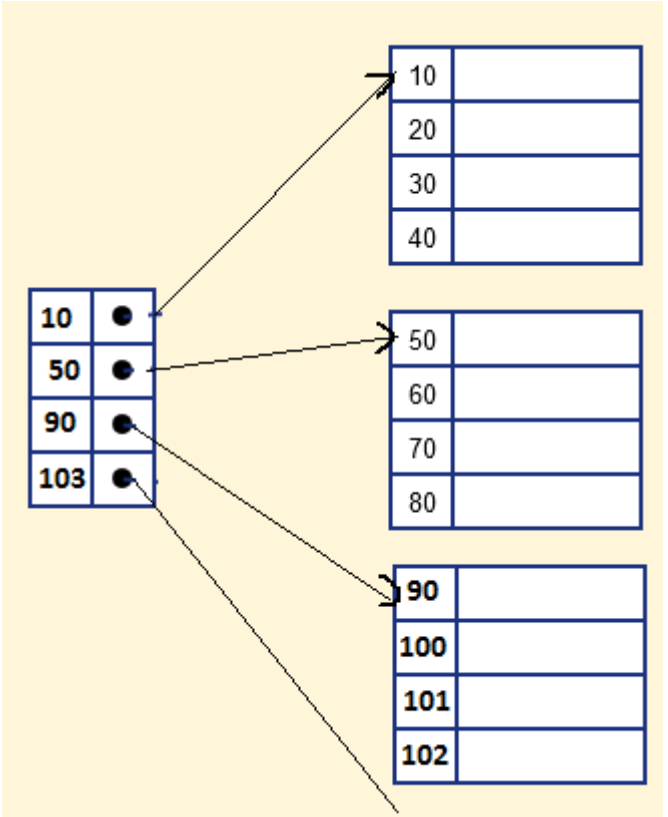
Dense Index

- In dense index, there is an index record for every search key value in the database.
- This makes searching faster but requires more space to store index records itself.



Sparse Index

- It is an index record that appears for only some of the values in the file.
- Sparse Index helps you to resolve the issues of dense Indexing in DBMS.
- In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.
- However, sparse Index stores index records for only some search-key values.
- It needs less space, less maintenance overhead for insertion, and deletions but It is slower compared to the dense Index for locating records



Primary index

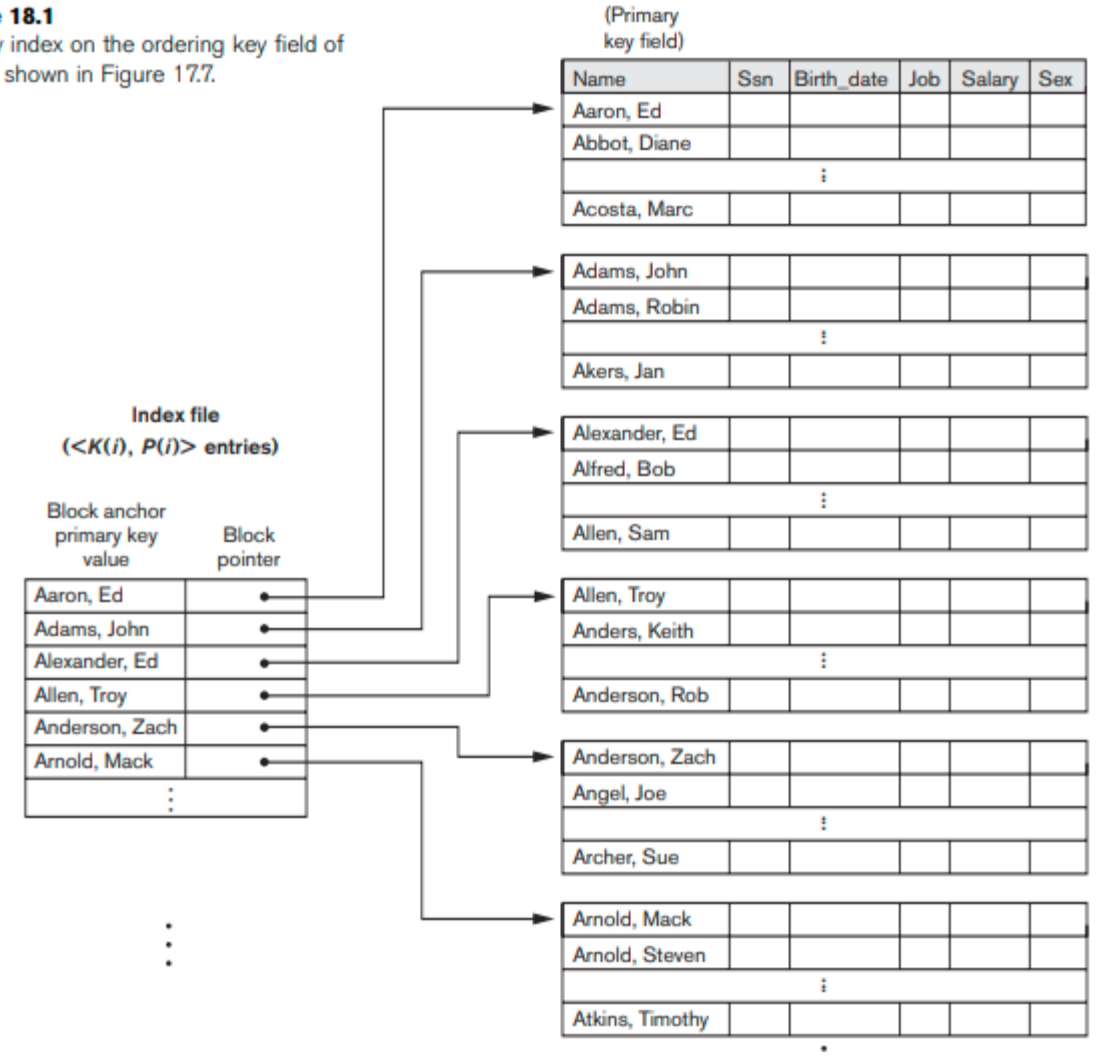
- A primary index is an ordered file whose records are of fixed length with two fields.
  - The first field is of the same data type as the ordering key field—called the primary key—of the data file,
  - and the second field is a pointer to a disk block (a block address).
- There is one index entry (or index record) in the index file for each block in the data file.
- Each index entry has the value of the primary key field for the first record in a block and a pointer to that block as its two field values.
- We will refer to the two field values of index entry i as <K(i),P(i)>

Example

- To create a primary index on the ordered, we use the Name field as primary key, because that is the ordering key field of the file (assuming that each value of Name is unique).
- Each entry in the index has a Name value and a pointer. The first three index entries are as follows:

<K(1) = (Aaron, Ed), P(1) = address of block 1>
<K(2) = (Adams, John), P(2) = address of block 2>
<K(3) = (Alexander, Ed), P(3) = address of block 3>

Figure 18.1
Primary index on the ordering key field of the file shown in Figure 17.7.



Cluster index

- Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them.
- This method is called a clustering index.
- A clustered index can be defined as an ordered data file.
- The records which have similar characteristics are grouped, and indexes are created for these group.

- Ordered file with two fields
  - Same type as clustering field
  - Disk block pointer

Difference between Cluster and Non-cluster Index

1. A clustered index defines the order in which data is **physically** stored in a table. For example Dictionary.

A non-clustered index is stored at one place and table data is stored in another place. For example Book Index.

2. A table can have only one clustered index.

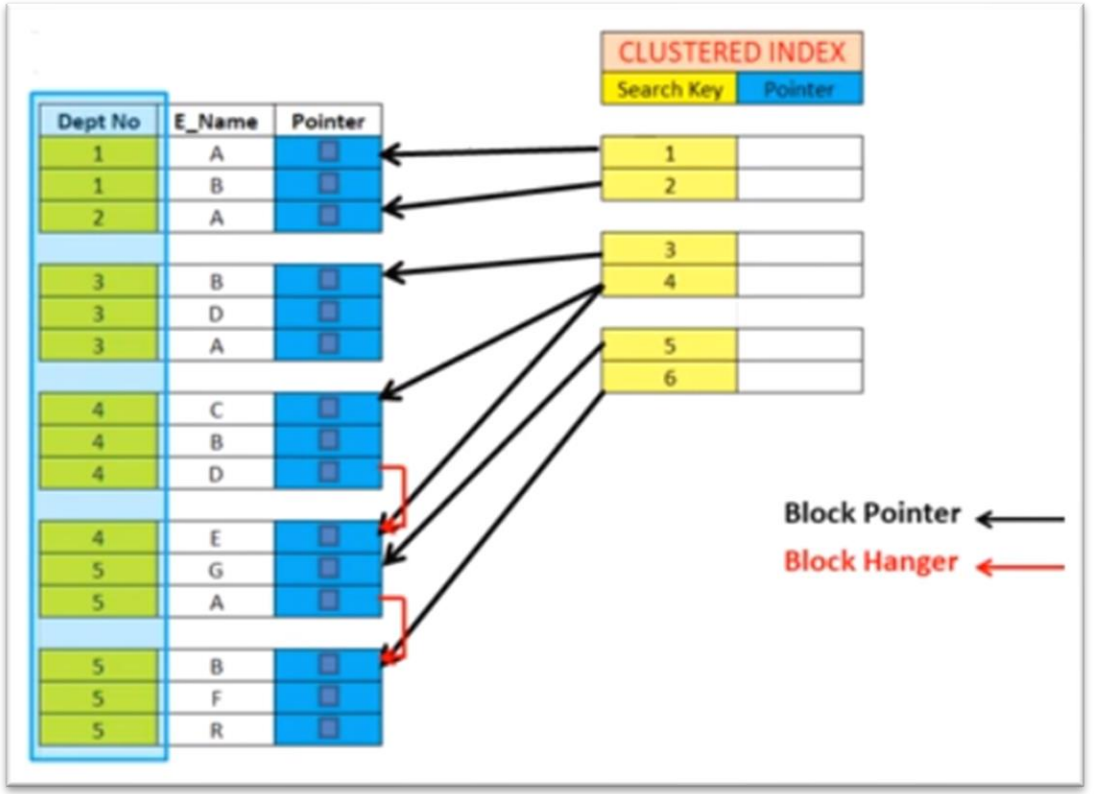
A table can have multiple non-clustered index.

3. Clustered index is faster.

Non-clustered index is slower.

Example


CLUSTERED INDEX
Search Key
Pointer
Dept No
E_Name
Pointer
1 A
1 B
2 A
3 B
3 D
3 A
4 C
4 B
4 D
4 E
5 G
5 A
5 B
5 F
5 R
1
2
3
4
5
6


CLUSTERED INDEX
Search Key
Pointer
Dept No
E_Name
Pointer
1 A
1 B
2 A
3 B
3 D
3 A
4 C
4 B
4 D
4 E
5 G
5 A
5 B
5 F
5 R
1
2
3
4
5
6
Block Pointer
Block Hanger

Secondary/Non-cluster Indexing

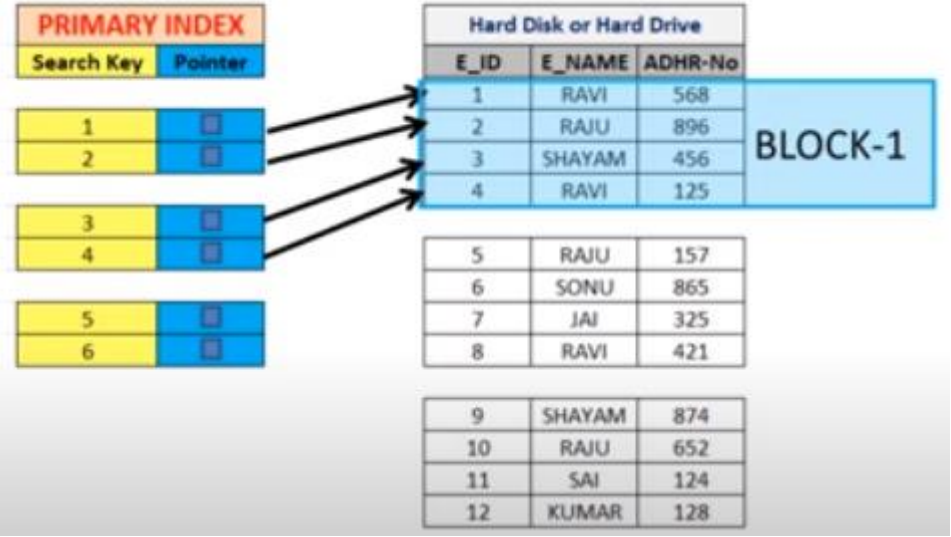
When to use Secondary Index ?

1. Unordered Data
2. Non-Key or Key based search required.
3. Data already have Primary Index

Cntd..

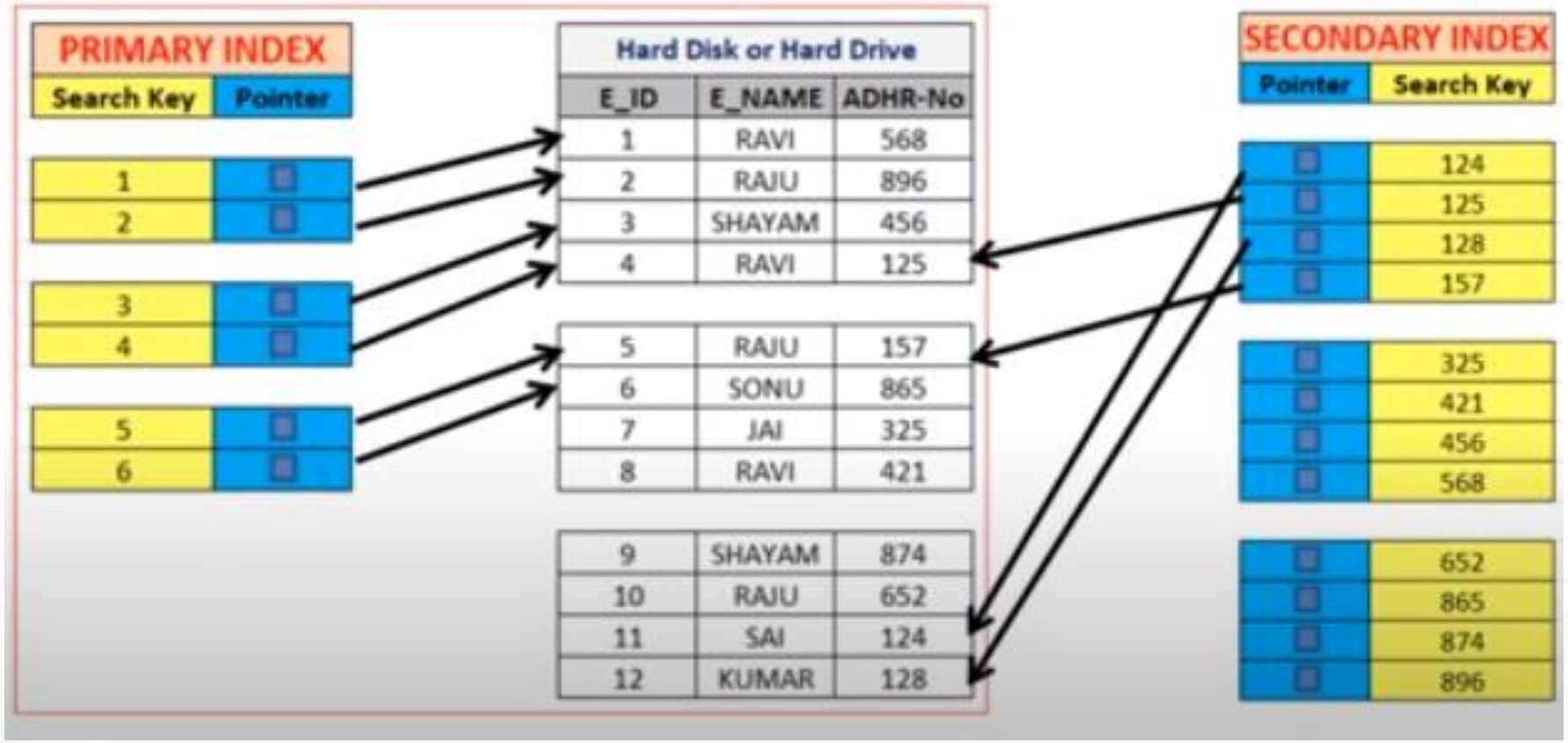
- The secondary Index in DBMS can be generated by a field which has a unique value for each record, and it should be a candidate key.
- It is also known as a non-clustering index.
- If the mapping size grows then fetching the address itself becomes slower.
- In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.
- In secondary indexing, to reduce the size of mapping, another level of indexing is introduced.
- In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small.
- Then each range is further divided into smaller ranges.
- The mapping of the first level is stored in the primary memory, so that address fetch is faster.
- The mapping of the second level and actual data are stored in the secondary memory (hard disk).

Example



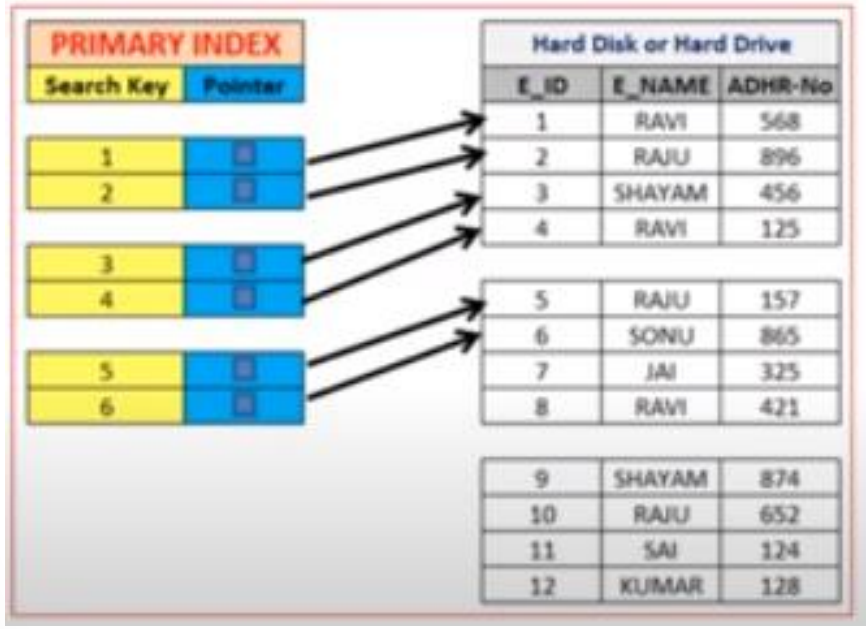
When you need to search data OTHER THAN KEY OF PRIMARY INDEX, such as Aadhar Card No, which is also a UNIQUE KEY IN DATABASE.

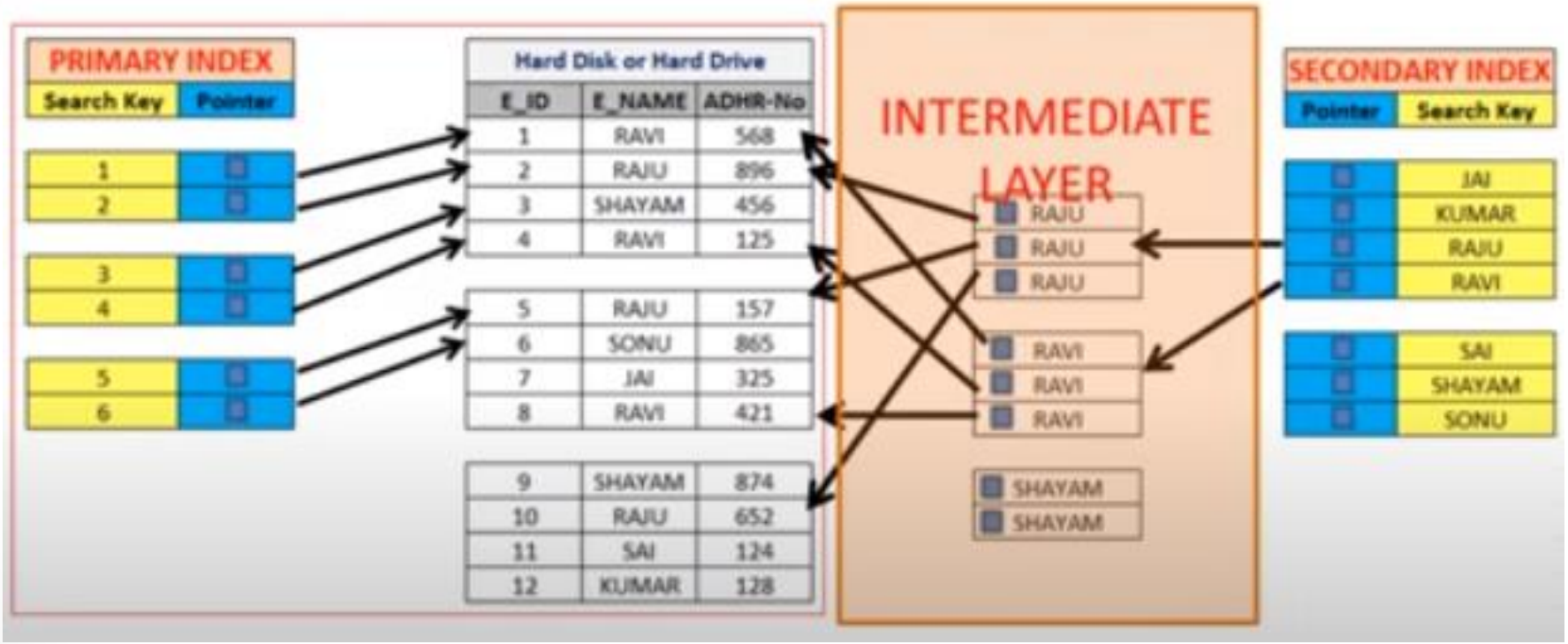
Secondary index on key-attribute



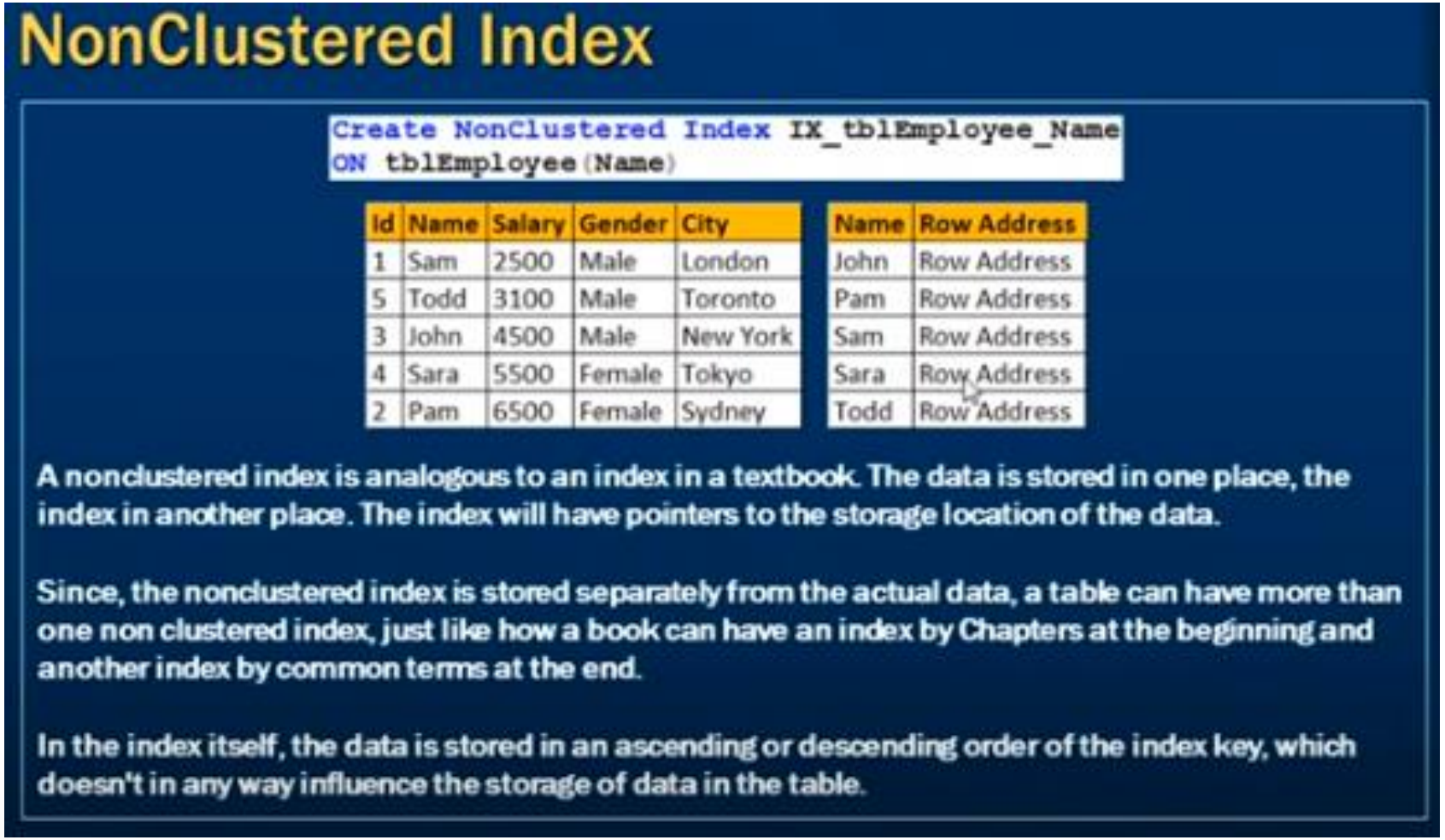
Secondary index on non-key attribute

When you need to search data OTHER THAN KEY OF PRIMARY INDEX, such as E_Name, which is NON KEY and having duplicate values in table of a DATABASE.




PRIMARY INDEX
Search Key
Pointer
1
2
3
4
5
6
Hard Disk or Hard Drive
E_ID
E_NAME
ADHR-No
1 RAVI 568
2 RAJU 896
3 SHAYAM 456
4 RAVI 125
5 RAJU 157
6 SONU 865
7 JAI 325
8 RAVI 421
9 SHAYAM 874
10 RAJU 652
11 SAI 124
12 KUMAR 128
INTERMEDIATE LAYER
RAJU
RAJU
RAJU
RAVI
RAVI
RAVI
SHAYAM
SHAYAM
SECONDARY INDEX
Pointer
Search Key
JAI
KUMAR
RAJU
RAVI
SAI
SHAYAM
SONU

NonClustered Index



Unique Index

When should you be creating a Unique constraint over a unique index?

To make our intentions clear, create a unique constraint, when data integrity is the objective. This makes the objective of the index very clear. In either cases, data is validated in the same manner, and the query optimizer does not differentiate between a unique index created by a unique constraint or manually created.

Useful points to remember:

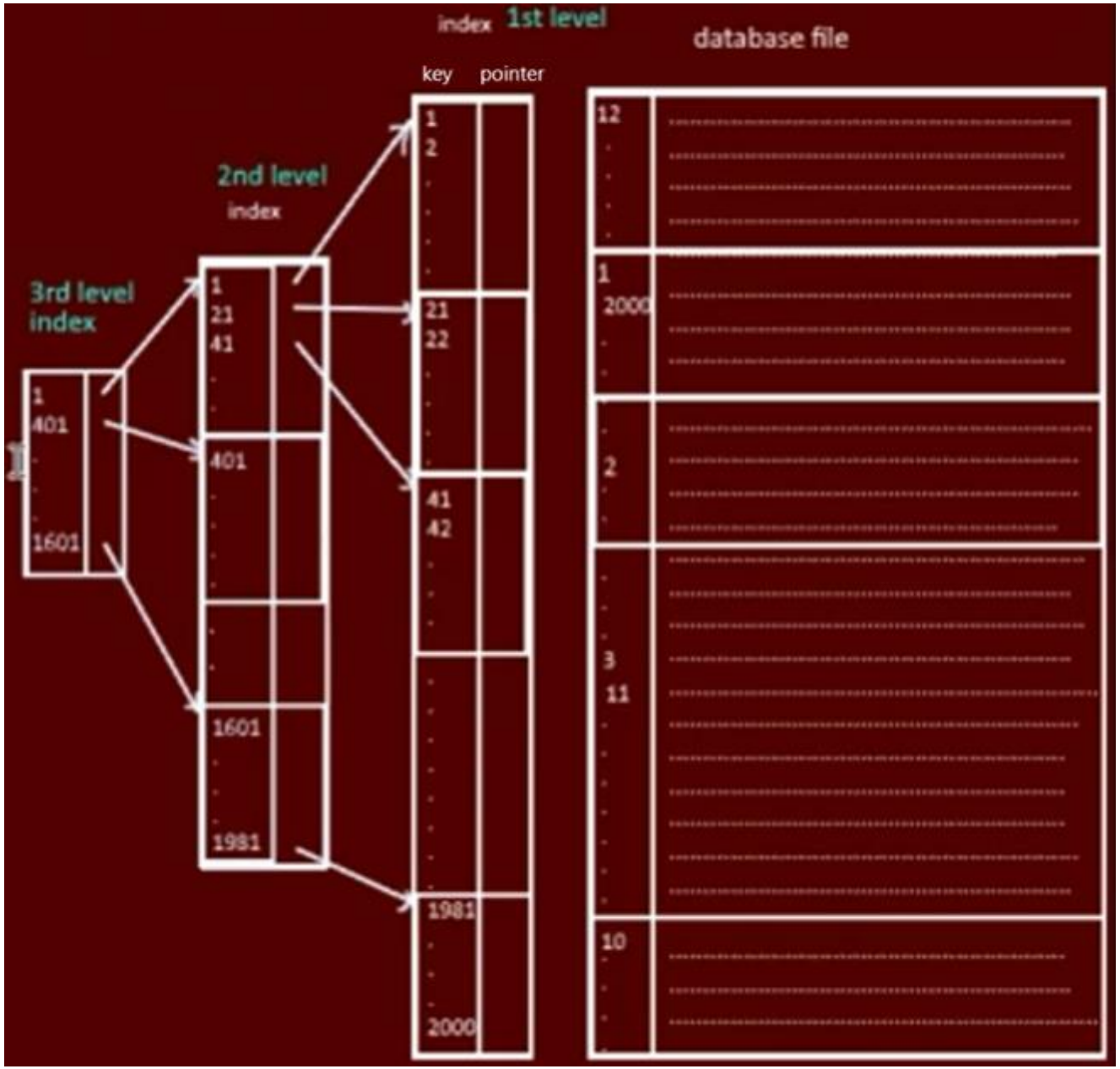
1. By default, a PRIMARY KEY constraint, creates a unique clustered index, where as a UNIQUE constraint creates a unique Non-Clustered index. These defaults can be changed if you wish to.

2. A UNIQUE constraint or a UNIQUE index cannot be created on an existing table, if the table contains duplicate values in the key columns. Obviously, to solve this, remove the key columns from the index definition or delete or update the duplicate values.

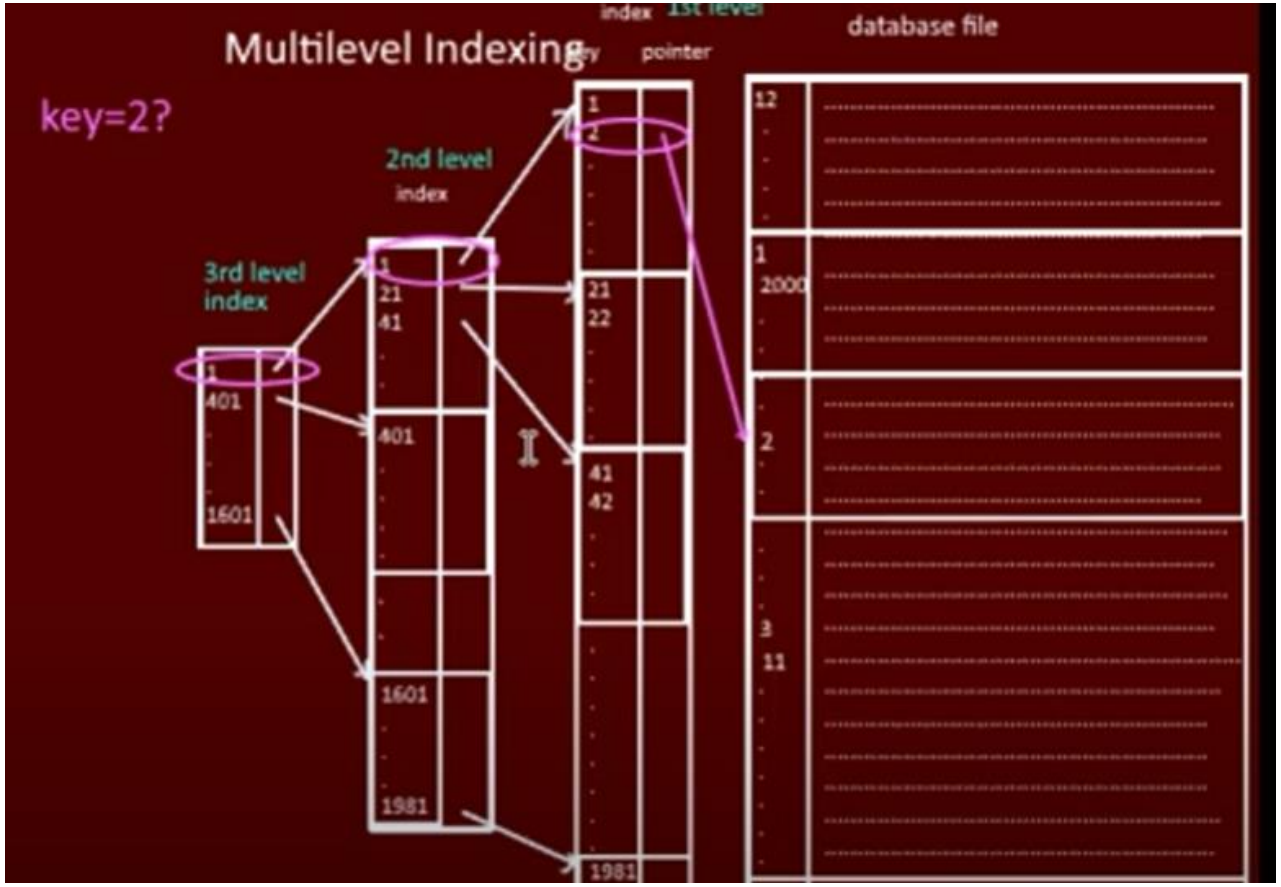
Multilevel index

- Index records comprise search-key values and data pointers.
- Multilevel index is stored on the disk along with the actual database files.
- As the size of the database grows, so does the size of the indices.
- There is an immense need to keep the index records in the main memory so as to speed up the search operations.
- If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.
- Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

Example

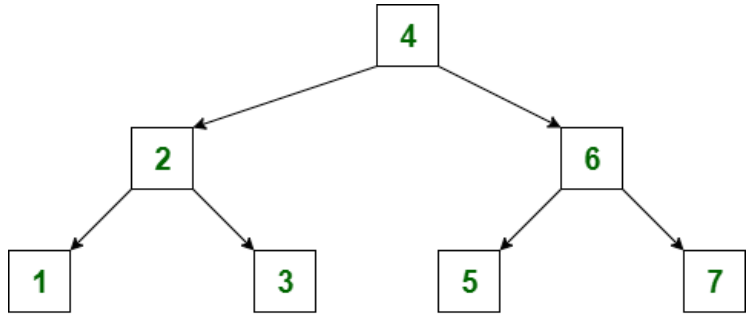

index 1st level
database file
key pointer
2nd level
index
3rd level
index
1
401
1601
1
21
41
401
1601
1981
1
2
21
22
41
42
1981
2000
12
1
2000
2
3
11
10

Searching in multilevel indexing



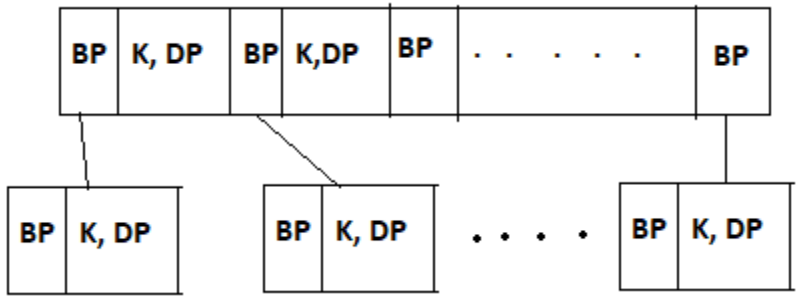
B-Tree

- B-Tree is known as a self-balancing tree as its nodes are sorted in the inorder traversal.
- In B-tree, a node can have more than two children.
- B-tree has a height of logM N (Where ‘M’ is the order of tree and N is the number of nodes).
- And the height is adjusted automatically at each update. In the B-tree data is sorted in a specific order, with the lowest value on the left and the highest value on the right.



B-Tree

- The structure of node in Btree is same at all level. (root,intemediate,leaf)
- BP-Block Pointer
- K-kev



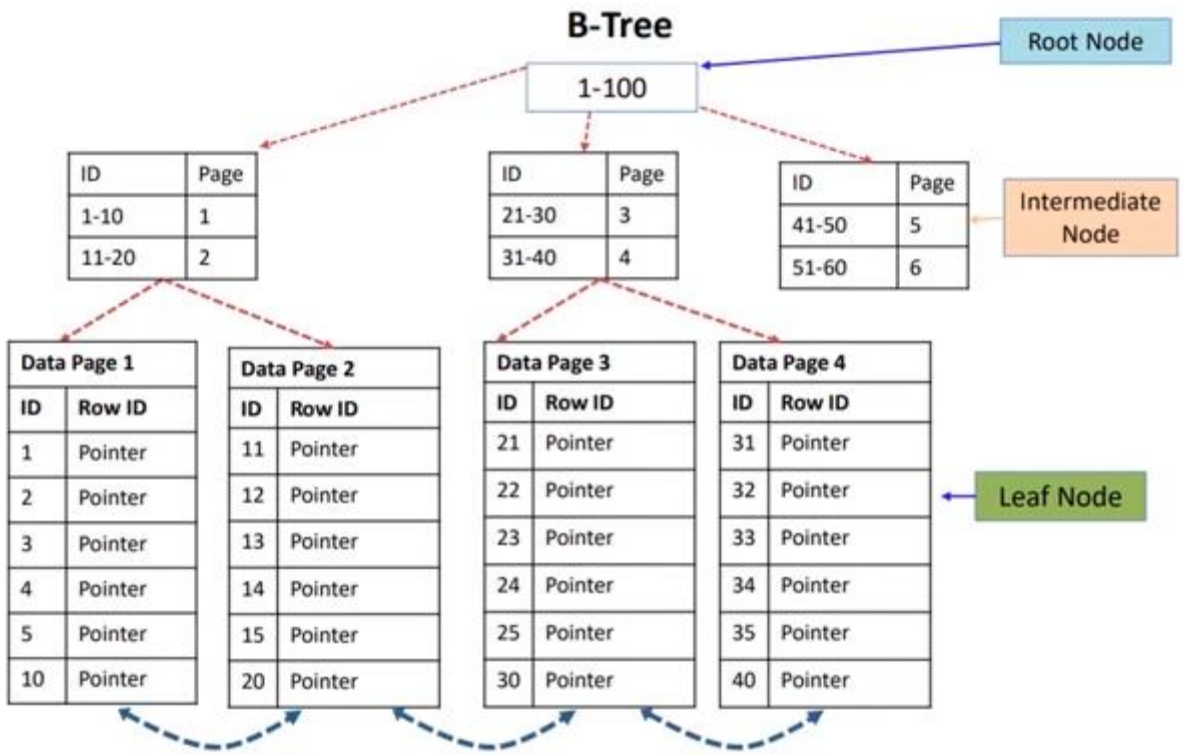
- Properties:
- If order of tree=m
- Max no.of children =m
- Min no.of keys=m-1
- Min no.of children = $\lceil m/2 \rceil$
- Min No. of keys= $\lceil m/2 \rceil - 1$

Example

- https://www.youtube.com/watch?v=KnXohGgIpQU

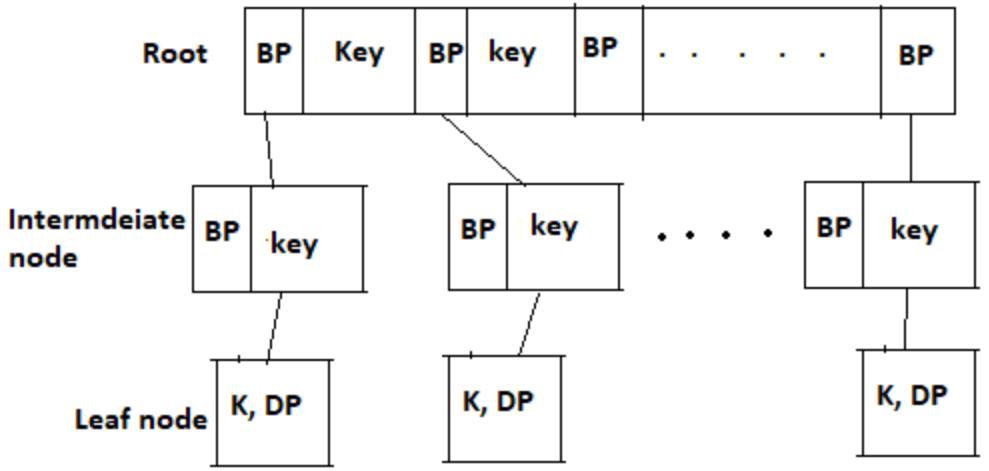
- https://www.youtube.com/watch?v=jpS8BLb8BgI

B-Tree


B-Tree
Root Node
1-100
ID
Page
1-10
1
11-20
2
21-30
3
31-40
4
41-50
5
51-60
6
Intermediate Node
Data Page 1
ID
Row ID
1
Pointer
2
Pointer
3
Pointer
4
Pointer
5
Pointer
10
Pointer
Data Page 2
11
12
13
14
15
20
Data Page 3
21
22
23
24
25
30
Data Page 4
31
32
33
34
35
40
Leaf Node

B^+ Tree

- A B^+ tree is a balanced binary search tree that follows a multi-level index format.
- The leaf nodes of a B^+ tree denote actual data pointers.
- B^+ tree ensures that all leaf nodes remain at the same height, thus balanced.
- Additionally, the leaf nodes are linked using a link list; therefore, a B^+ tree can support random access as well as sequential access.
-



Create index

- Syntax:

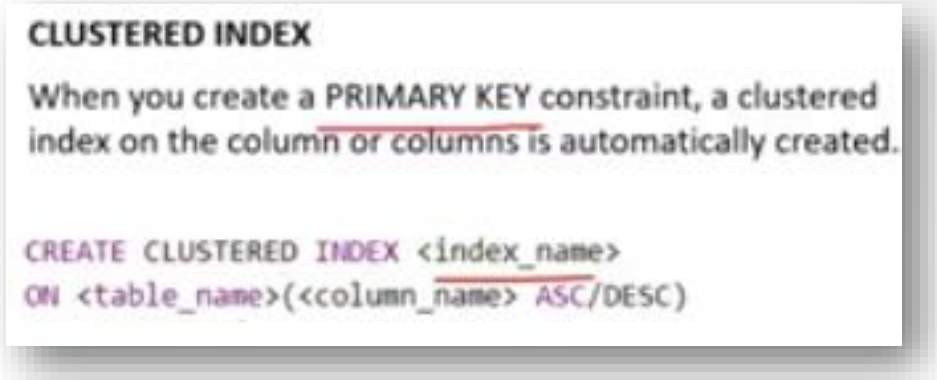
  CREATE INDEX *index_name*
  ON *table_name* (*column1*, *column2*, ...);

- CREATE UNIQUE INDEX
- Duplicate values are not allowed.
- Syntax:

  CREATE UNIQUE INDEX *index_name*
  ON *table_name* (*column1*, *column2*, ...);

- DROP INDEX
- The DROP INDEX statement is used to delete an index in a table.

  ALTER TABLE *table_name* DROP INDEX *index_name*;



Clustered index

A clustered index determines the physical order of data in a table. For this reason, a table can have only one clustered index.

```
CREATE TABLE [tblEmployee]
(
  [Id] int Primary Key,
  [Name] nvarchar(50),
  [Salary] int,
  [Gender] nvarchar(10),
  [City] nvarchar(50)
)
```

Note that Id column is marked as primary key. Primary key, constraint create clustered indexes automatically if no clustered index already exists on the table.

To confirm: `Execute sp_helpindex tblEmployee`

Index Example

At the moment, the Employees table, does not have an index on SALARY column.

Id	Name	Salary	Gender
1	Sam	2500	Male
2	Pam	6500	Female
3	John	4500	Male
4	Sara	5500	Female
5	Todd	3100	Male

```
Select * from tblEmployee
where Salary > 5000 and Salary < 7000
```

To find all the employees, who has salary greater than 5000 and less than 7000, the query engine has to check each and every row in the table, resulting in a table scan, which can adversely affect the performance, especially if the table is large. Since there is no index, to help the query, the query engine performs an entire table scan.

Creating an Index

```
CREATE Index IX_tblEmployee_Salary
ON tblEmployee (SALARY ASC)
```

The index stores salary of each employee, in the ascending order as shown below. The actual index may look slightly different.

Id	Name	Salary	Gender
1	Sam	2500	Male
2	Pam	6500	Female
3	John	4500	Male
4	Sara	5500	Female
5	Todd	3100	Male

Salary	RowAddress
2500	Row Address
3100	Row Address
4500	Row Address
5500	Row Address
6500	Row Address

Now, when the SQL server has to execute the same query, it has an index on the salary column to help this query. Salaries between the range of 5000 and 7000 are usually present at the bottom, since the salaries are arranged in an ascending order. SQL server picks up the row addresses from the index and directly fetch the records from the table, rather than scanning each row in the table. This is called as Index Seek.

Example

```
SQLQuery2.sql - (L...).Sample (sa (56))*
Select * from tblEmployee

Create Index IX_tblEmployee_Salary
ON tblEmployee (Salary ASC)
```

```
sp_Helpindex tblEmployee

drop index tblEmployee.IX_tblEmployee_Salary
```

Cluster index

```
SQLQuery6.sql - (L...).Sample (sa (54))*  SQLQuery5.sql - (L...).Sample (sa (52))*
CREATE TABLE [tblEmployee]
(
    [Id] int Primary Key,
    [Name] nvarchar(50),
    [Salary] int,
    [Gender] nvarchar(10),
    [City] nvarchar(50)
)

Execute sp_helpindex tblEmployee
```