CO-3

CO3 Topics

- PL/SQL
- Normalization
- Database File Organization
- Database File indexing
- Algorithms for PROJECT and Set Operations
- Implementing Aggregate Operations and OUTER JOINs

PL/SQL

- Introduction
- Features of PL/SQL
- PL/SQL Structure
- Programming constructs

Introduction to PL/SQL

- What is PL/SQL?
  - Developed by Oracle corporation in the 1980s.
  - It's a procedure language extension for SQL and Oracle relational database.

Features of PL/SQL

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types as SQL.
- It offers a variety of programming structures
- It supports object-oriented programming.
- It supports the development of web applications and server pages.
- It is not case sensitive.

PL/SQL structure

```
DECLARE
    <declarations section>
BEGIN
    <executable command(s)>
EXCEPTION
    <exception handling>
END;
```

The 'Hello World' Example

```
DECLARE
        message varchar2(20) := ‘Hello World!’;
BEGIN
        dbms_output.put_line(message);
END;
```

Output:

```
Hello World!

PL/SQL procedure successfully completed.
```

Programming structures

- Conditional statements
- Loops
- Procedures
- Functions
- Cursors
- Triggers

MySQL
Stored Procedures

MySQL Stored Procedures

Sid	Sname	Age	Dept	Year	Mobile	Address	CGPA

- **SELECT** Sid, Dept, Year, CGPA **From** students;
- MySQL processes the query and returns the result set.
- If you want to save this query on the database server for execution later, one way is to use a stored procedure.
- Once you save the stored procedure, you can invoke it by using the CALL statement:

MySQL Stored Procedures…

- A Stored procedure is a subroutine which contains a set of statements, stored in database.
- A procedure has a name, a parameter list, and SQL statement(s).
- It does not return a value.

Syntax to create procedure

```
CREATE PROCEDURE procedure_name [ (parameter_type    parameter    datatype
                                  [, parameter_type    parameter    datatype ]) ]
BEGIN

    declaration_section

    executable_section

END;
```

Syntax for calling Procedure:

```
CALL procedure_name([parameters]);
```

Example

```
CREATE PROCEDURE GetStudents( )

BEGIN

  SELECT Sid, Dept, Year, CGPA FROM students;

END
```

```
CALL GetStudents( );
```

DROP Procedure

- Once you have created your procedure in MySQL, you might find that you need to remove it from the database.

- The syntax to a drop a procedure in MySQL is:

  DROP procedure [IF EXISTS] ***procedure_name*****;**

Steps to follow before creating a procedure

1. Select a database
2. Pick a Delimiter

Examples

- Simple procedure

```
delimiter @
create procedure add1()
begin
 declare a, b, c int;

 set a=10;
 set b=20;
 set c=a+b;
 select c;
end @
delimiter ;

call add();

drop procedure add1;
```

Examples

- Procedure with parameters

```
delimiter @
create procedure add2(in a int, in b int)
begin
 declare c int;

 set c=a+b;
 select c;
end @
delimiter ;

call add2(10,20);

drop procedure add2;
```

IF-THEN-ELSE Statement

IF-THEN-ELSE Statement

```
IF condition1 THEN
  {...statements to execute when condition1 is TRUE...}

[ ELSEIF condition2 THEN
  {...statements to execute when condition1 is FALSE and condition2 is TRUE...}]

[ ELSE
  {...statements to execute when both condition1 and condition2 are FALSE...}]

END IF;
```

Examples

- Simple IF-ELSE

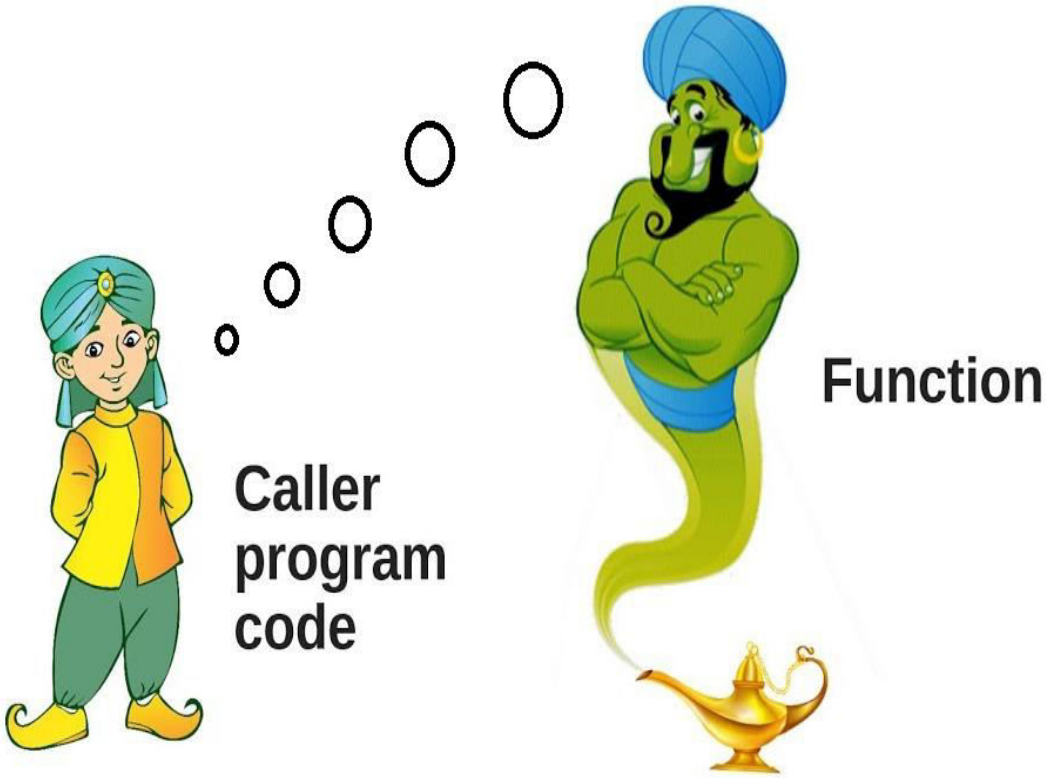
```
drop procedure if exists set_b;
delimiter @
create procedure set_b(in a int, out b int)
begin

   declare c int;
   set c = 100;
   if a < 20 then
      set b = c + a;
   else
      set b = c - a;
   end if;

end @
delimiter ;

call set_b(30, @b);
select @b;
```

Examples

- IF-ELSEIF-ELSE

```
drop procedure if exists grade_result;
delimiter @
create procedure grade_result(in p int)
begin
   if p >= 90 then
      select 'Grade : X';
   elseif p >= 75 and p < 90 then
      select 'Grade : A';
   elseif p >= 60 and p < 75 then
      select 'Grade : B';
   elseif p >= 40 and p < 60 then
      select 'Grade : C';
   else
      select 'Grade : Fail';
   end if;
end @
delimiter ;

call grade_result(80);
```

MySQL Functions



MySQL Functions

- In MySQL, Functions can also be created.
- Similar to a procedure.
- A function always **returns a value** using the return statement.
- Function call is done using **SELECT.**

Function Syntax

Creating a function

```
CREATE FUNCTION function_name [ (parameter datatype
                                 [, parameter datatype]) ]
RETURNS return_datatype
BEGIN
        Declaration_section
        Executable_section
END;
```

Drop a function

```
DROP FUNCTION [ IF EXISTS ] function_name;
```

Example

- Simple function

```
drop function if exists grade_result;
delimiter @
create function grade_result(p int) returns varchar(10)
begin
   declare grade varchar(10);
   if p >= 90 then
      set grade = 'X';
   elseif p >= 75 and p < 90 then
      set grade = 'A';
   elseif p >= 60 and p < 75 then
      set grade = 'B';
   elseif p >= 40 and p < 60 then
      set grade = 'C';
   else
      set grade = 'Fail';
   end if;
   return grade;
end @
delimiter ;

select grade_result(80);
```

CASE Statements

CASE Statement

```
CASE [ expression ]
        WHEN condition_1 THEN result_1
        WHEN condition_2 THEN result_2   ...
        WHEN condition_n THEN result_n
        ELSE result
END
```

Note

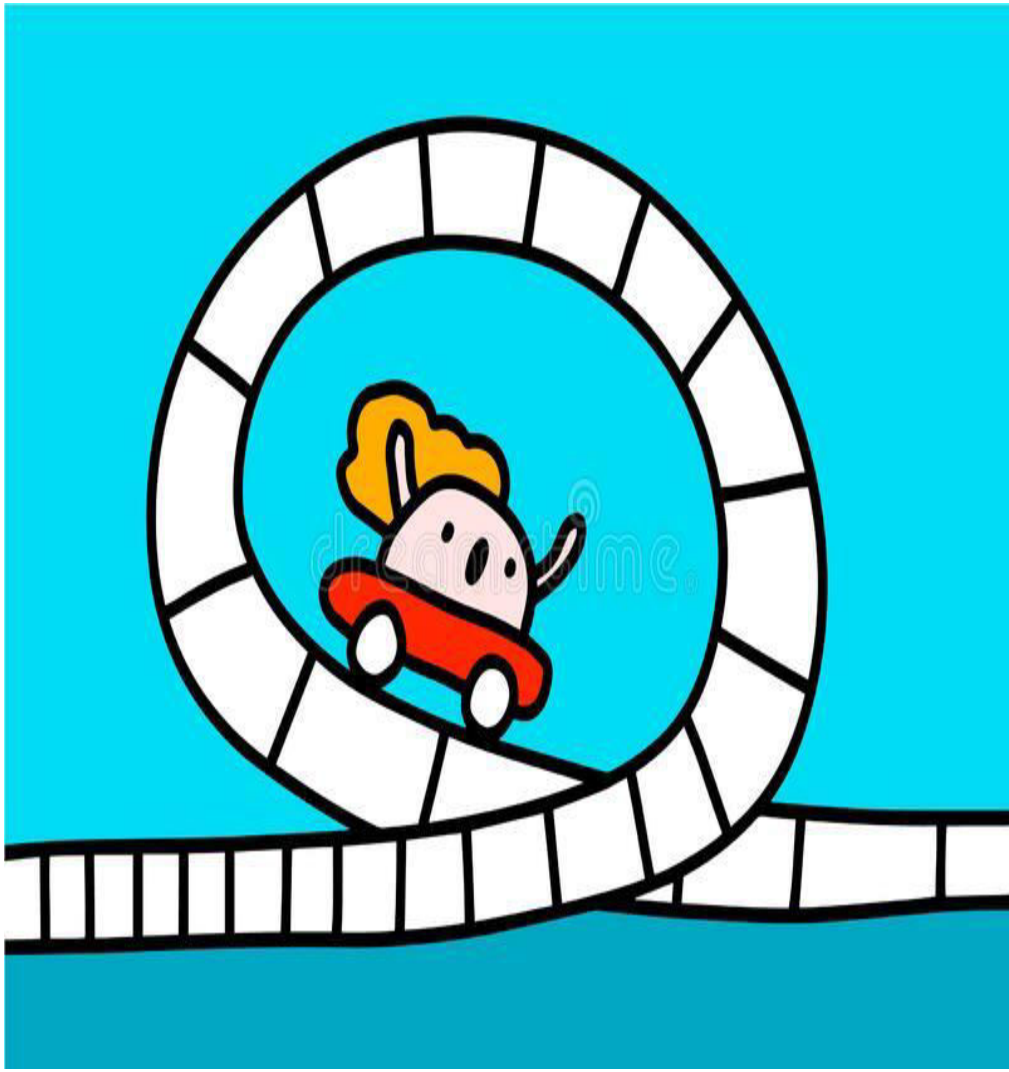
- If no condition is found to be true, then the CASE function will return the value in the ELSE clause.
- If the ELSE clause is omitted and no condition is found to be true, then the CASE statement will return NULL.

Example

- Simple CASE

```
drop function if exists grade;
delimiter @
create function grade(p int) returns varchar(10)
begin
   declare grade varchar(10);
   case
   when p >= 90 then
      set grade = 'X';
   when p >= 75 and p < 90 then
      set grade = 'A';
   when p >= 60 and p < 75 then
      set grade = 'B';
   when p >= 40 and p < 60 then
      set grade = 'C';
   else
      set grade = 'Fail';
   end case;
   return grade;
end @
delimiter ;

select grade(80);
```

LOOP Statements



MySQL Loops

- LOOP
  - allows you to execute one or more statements repeatedly.
- LEAVE
  - exits the flow control that has a given label.
- REPEAT
  - executes one or more statements until a search condition is true.
- WHILE
  - executes a block of code repeatedly as long as a condition is true.

Syntax of MySQL Loops

LOOP	LEAVE
[begin_label:] **LOOP** statement_list **END LOOP** [end_label]	**LEAVE** label;
REPEAT	**WHILE**
[begin_label:] **REPEAT** statement_list **UNTIL** search_condition **END REPEAT** [end_label]	[begin_label:] **WHILE** search_condition **DO** statement_list **END WHILE** [end_label]

Examples

- LOOP and LEAVE

```
drop procedure if exists mulofn;
delimiter @
create procedure mulofn(a int)
begin

   declare c,i int;
   set i=1;
   L1: loop
        set c = i * a;
        select a,i,c;
        set i = i + 1;
        if i<=10 then
           iterate L1;
        end if;
        leave L1;
        end loop L1;

   end @
delimiter ;

call mulofn(10);
```

Examples

- REPEAT

```
drop function if exists sumofn;
delimiter @
create function sumofn(n int) returns int
begin

   declare sum,i int;
   set sum = 0, i = 1;
   L1: repeat
        set sum = sum + i;
        set i = i + 1;
        until i>n
        end repeat L1;
   return sum;
   end @
delimiter ;

call mulofn(5);
```

Examples

- WHILE

```
drop procedure if exists mulofn;
delimiter @
create procedure mulofn(a int)
begin

   declare c,i int;
   set i=1;
   L1: while i<=10 do
        set c = i * a;
        select a,i,c;
        set i = i + 1;
   end while L1;

end @
delimiter ;

call mulofn(10);
```

Example

- To extract data from tables

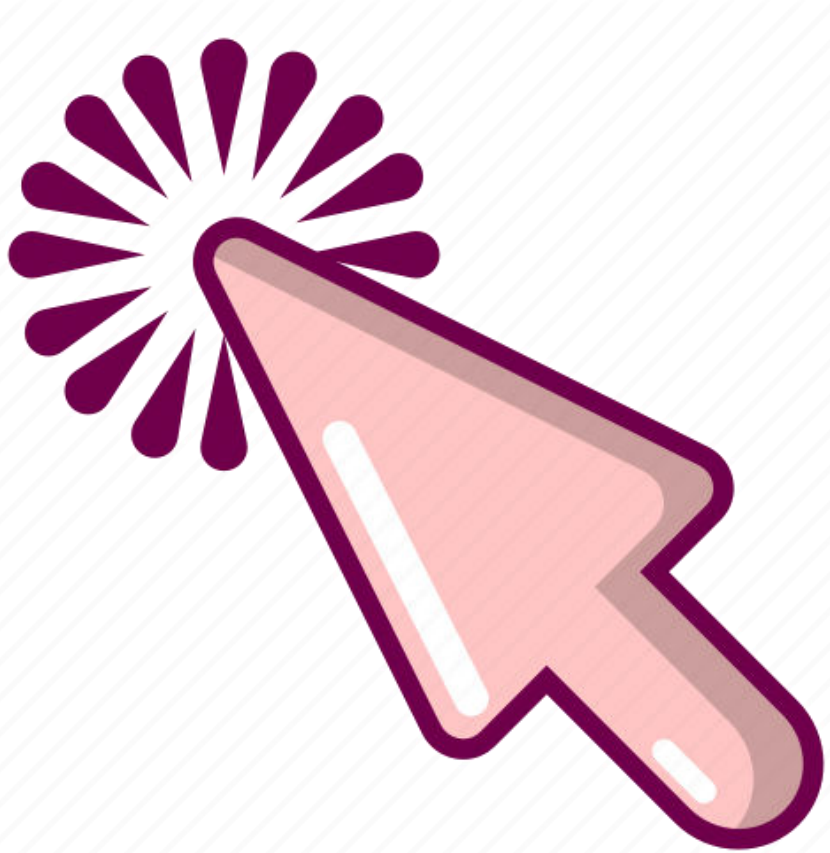
```
drop function if exists grade;
delimiter @
create function grade(id int) returns varchar(10)
begin
   declare grade varchar(10);
   declare s1,s2,s3,sum int;
   declare p float;
   select m1 into s1 from students where sno = id;
   select m2 into s2 from students where sno = id;
   select m3 into s3 from students where sno = id;
   set sum = s1 + s2 + s3;
   set p = (sum / 90)*100;
   case
   when p >= 90 then
      set grade = 'X';
   when p >= 75 and p < 90 then
      set grade = 'A';
   when p >= 60 and p < 75 then
      set grade = 'B';
   when p >= 40 and p < 60 then
      set grade = 'C';
   else
      set grade = 'Fail';
   end case;
   return grade;
end @
delimiter ;
```

Session-13

Recap

- Features of PL/SQL
- MySQLProcedures
- MySQL Functions
- Conditional Statements
- MySQL Loops

MySQL CURSORS



Introduction to MySQL Cursor

- To handle a result set inside a stored procedure, you use a cursor.

- A cursor allows you to iterate a set of rows returned by a query and process each row individually.

- In simple, a cursor is a mechanism by which you can assign a name to a SELECT statement.

- MySQL cursor is:
  - **Read-only**: Not updatable
  - **Non-scrollable**: Can be traversed only in one direction and cannot skip rows
  - **Asensitive**: The server may or may not make a copy of its result table

- You can use MySQL cursors in stored procedures, stored functions, and triggers.

Working with MySQL cursor

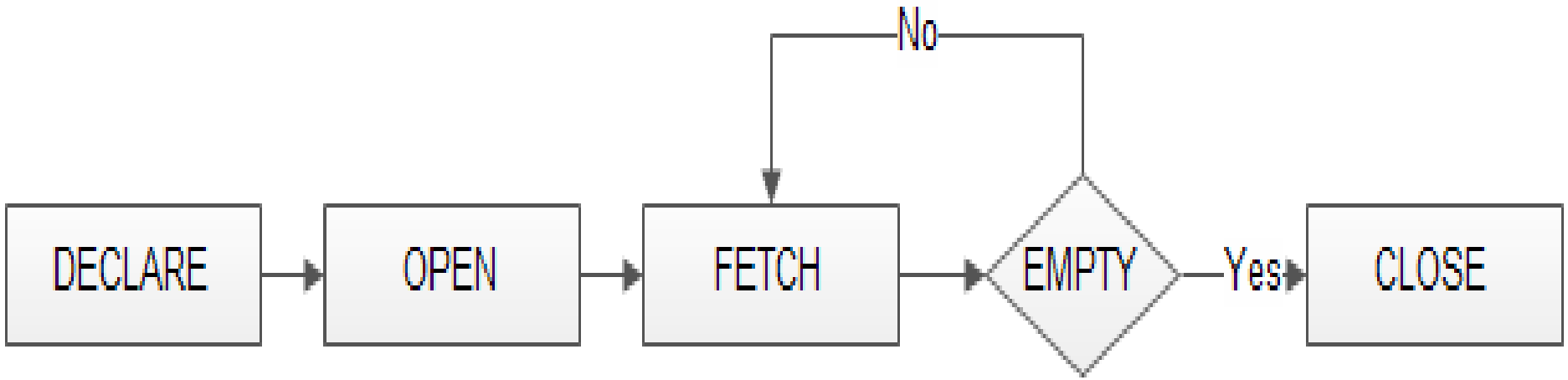
- To create a cursor, we need to perform the following steps.

  - DECLARE Cursor
  - OPEN Cursor
  - FETCH Cursor
  - CLOSE Cursor
  - Set up Handler for Cursor's NOT FOUND condition

Introduction to MySQL Cursor...

1. Declare Cursor	2. Open Cursor
DECLARE cursor_name CURSOR FOR select_statement;	OPEN cursor_name;
3. Fetch Cursor	**4. Close Cursor**
FETCH cursor_name INTO variables list;	CLOSE cursor_name;
5. NOT FOUND Handler	
DECLARE CONTINUE HANDLER FOR NOT FOUND [set_condition];	

Cursor Functioning



Example

- Cursor to get the list of Employee mail_ids

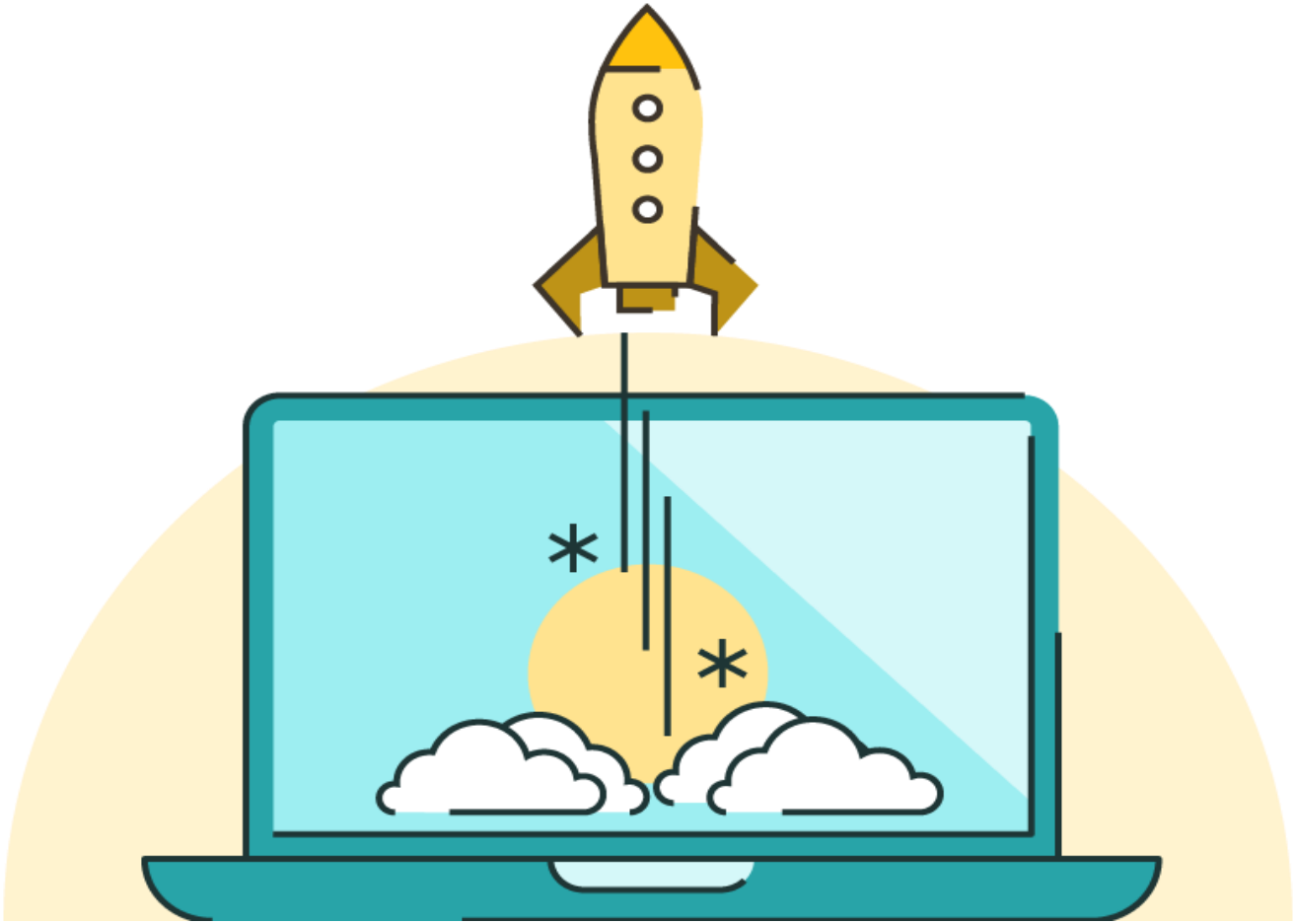
EID	ENAME	DEPT	DESG	MAID_ID
1	..	..	..	..
2	..	..	..	..

Example

- Cursor to get the list of Employee mail_ids

```
drop procedure if exists get_mailList;
delimiter @
create procedure get_mailList(inout mail_idList varchar(500))
begin
    declare email varchar(30) default "";
    declare finished int default 0;

    declare curEmail cursor for select mail_id from employee where dept = 'CSE';

    declare continue handler for not found set finished = 1;

    open curEmail;

    getmail: Loop
            fetch curEmail into email;
            if finished = 1 then
                leave getmail;
            end if;
            set mail_idList = concat(email, " ; ", mail_idList);
            end loop getmail;

    close curEmail;

end @;
delimiter ;
```

MySQL Triggers



Introduction to Triggers

- A trigger is a stored program invoked automatically in response to an event that occurs in the associated table.
- MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event.
- The SQL standard defines two types of triggers:
  - row-level triggers
  - statement-level triggers.
- MySQL supports only row-level triggers. It doesn't support statement-level triggers.

Syntax for Creating & Dropping Trigger

```
CREATE
        TRIGGER trigger_name
        trigger_time  trigger_event
        ON table_name FOR EACH ROW
        BEGIN
                trigger_body
        END
```

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

```
DROP TRIGGER trigger_name;
```

Types of Triggers

- There are 6 type of triggers that can be created.
  - Before Insert
  - After Insert
  - Before Update
  - After Update
  - Before Delete
  - After Delete

Value Modifiers

- The trigger body can access the values of the column being affected by the DML statement.
- To distinguish between the value of the columns BEFORE and AFTER the DML has fired, we use the **NEW** and **OLD** modifiers.

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Examples

- Simple triggers on EMPLOYEE table

```
AFTER INSERT
------------

create table employee_log (empid int, empname varchar(20), actiontype varchar(10), actiondate date);

drop trigger aft_emp_insert;
delimiter $$
create trigger aft_emp_insert after insert on employee for each row
begin
     insert into employee_log values(new.eid, new.ename, 'inserted', now());
end $$
delimiter ;


insert into employee values (9, 'Suma', 'ECM', 'HOD', 'suma@gmail.com');
select * from employee;
select * from employee_log;
```

Examples

- Simple triggers on EMPLOYEE table

```
BEFORE DELETE
-------------
drop trigger bef_emp_delete;
delimiter $$
create trigger bef_emp_delete before delete on employee for each row
begin
     insert into employee_log values(old.eid, old.ename, 'deleted', now());
end $$
delimiter ;

delete from employee where eid = 8;
```

Examples

- Simple triggers on EMPLOYEE table

```
/* BEFORE UPDATE on same Table */

drop trigger bef_emp_update;
delimiter $$
create trigger bef_emp_update before update on employee for each row
begin
    set new.ename = UPPER(new.ename);
end $$
delimiter ;

update employee set dept = 'ME' where eid = 8;
```

SUMMARY

- MyQSL Procedures
- MySQL Functions
- IF-THEN-ELSE Statements
- CASE Statements
- MySQL LOOPs
- MySQL Cursors
- MySQL Triggers

DATABASE NORMALIZATION

Radhika Rani Chintala

Start with Understanding

- Problems without Normalization
- How Normalization solves these problems

Radhika Rani Chintala

What is Normalization ?

- A technique of organizing the data into multiple related tables, to minimize **DATA REDUNDANCY.**

Radhika Rani Chintala

What is Data Redundancy ?

Why should we reduce it?

- Repetition of similar data at multiple places.

- Repetition of data increases the size of database.
- Other issues:
  - Insertion problems
  - Deletion problems
  - Updation problems

Example

Students Table

Sid	Sname	Branch	HOD	Office_Tel
1	Anandi	CSE	Mr. X	3031
2	Jaya	CSE	Mr. X	3031
3	Radhi	CSE	Mr. X	3031
4	Satya	CSE	Mr. X	3031

Branch name, HOD name and Office telephone number are same for all the rows and hence they are **redundant data**.

Issues due to Data Redundancy

- Increases size of database
- Insertion Anomaly
- Deletion Anomaly
- Updation Anomaly

Radhika Rani Chintala

Insertion Anomaly

Sid	Sname	Branch	HOD	Office_Tel
1	Anandi	CSE	Mr. X	3031
2	Jaya	CSE	Mr. X	3031
3	Radhi	CSE	Mr. X	3031
4	Satya	CSE	Mr. X	3031
5				
.				
.				
100				

To add a new student, we have to repeat the same data that holds the branch information(Branch, HOD, Office_Tel)

Radhika Rani Chintala

Insertion Anomaly

- Inserting redundant data for new row is called Data Insertion problem or anomaly

- **Reason for data repetition:**
  - Two different but related data is stored in the same table (Student data and Branch data)

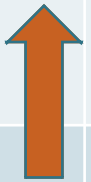
Radhika Rani Chintala

Deletion Anomaly

- When we try to delete student information(row), the branch information is also deleted

Deletion Anomaly

Sid	Sname	Branch	HOD	Office_Tel
1	Anandi	CSE	Mr. X	3031



We have not stored the branch information anywhere else.

When we delete the last row, We lose the branch information completely along with student information.

Radhika Rani Chintala

Deletion Anomaly

- Loss of related dataset when some other dataset is deleted is known as Deletion anomaly.

Updation Anomaly

Sid	Sname	Branch	HOD	Office_Tel
1	Anandi	CSE	Mr. X	3031
2	Jaya	CSE	Mr. X	3031
3	Radhi	CSE	Mr. X	3031
4	Satya	CSE	Mr. X	3031
5	Rani	CSE	Mr. X	3031

Mr. X leaves and **Mr. Y** joins as the new HOD for CSE, then you need to modify each row with new HOD

Radhika Rani Chintala

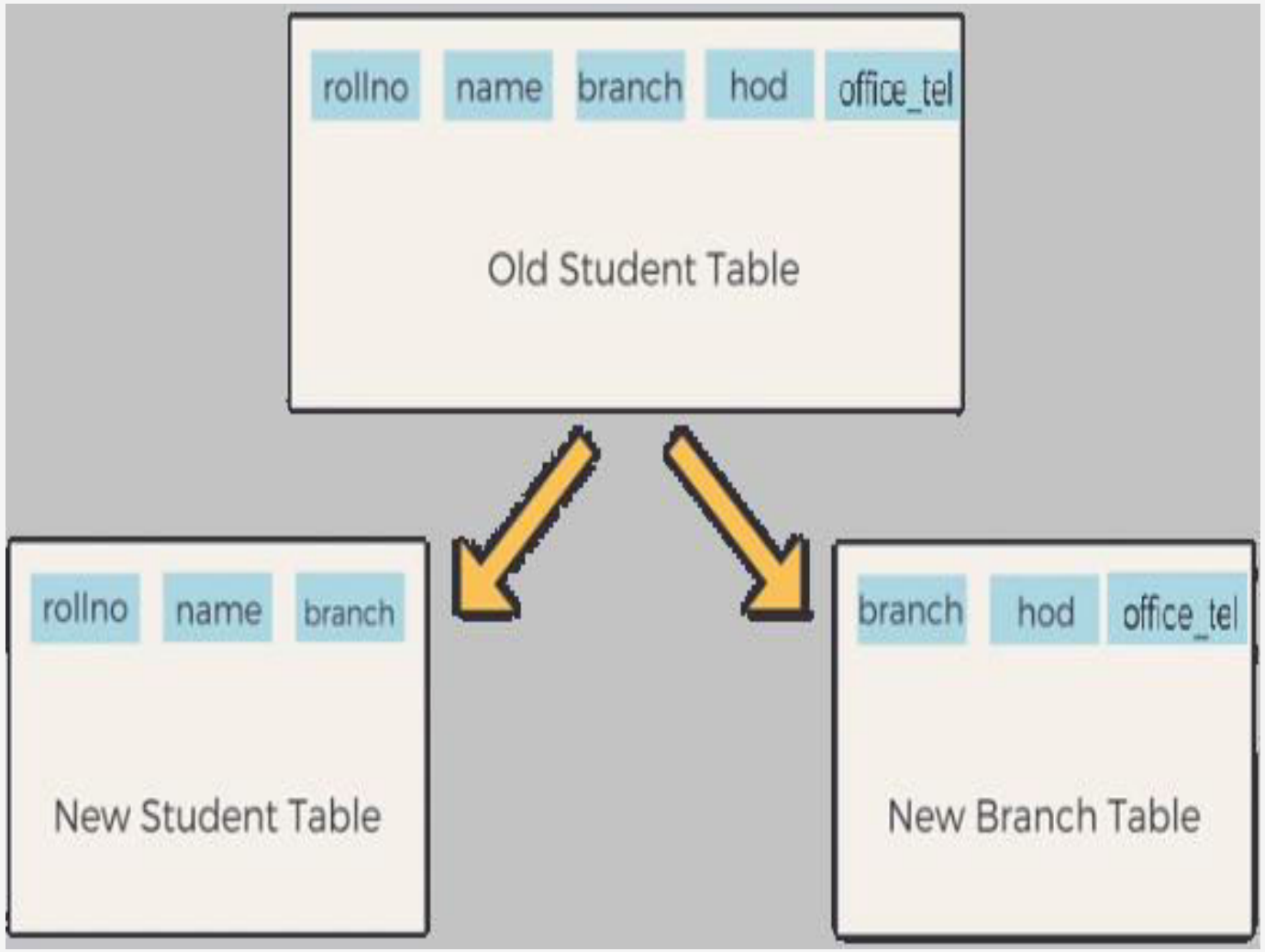
Data Redundancy

- Repetition of data hence needs extra space
- Leads to insertion, deletion and updation issues

Radhika Rani Chintala

How normalization will solve all these problems

???



Radhika Rani Chintala

Redundancy is reduced and thus reducing the problems of *insertion,deletion and updation*

Sid	Sname	Branch	HOD	Office_Tel
1	Anandi	CSE	Mr. X	3031
2	Jaya	CSE	Mr. X	3031
3	Radhi	CSE	Mr. X	3031

STUDENT TABLE

Sid	Sname	Branch
1	Anandi	CSE
2	Jaya	CSE
3	Radhi	CSE

BRANCH TABLE

Branch	HOD	Office_Tel
CSE	Mr. X	3031

Normalization is Good

- It follows Divide and Rule
- Divides the data into separate independent logical entities and relating them using some common entity(Branch)

Types of Normalization

- 1st Normal Form(1NF)
- 2nd Normal Form(2NF)
- 3rd Normal Form(3NF)
- Boyce-Codd Normal Form (BCNF)
- 4th Normal Form(4NF)
- 5th Normal Form(5NF)

Radhika Rani Chintala

1st Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. *It should only have single(atomic) valued attributes/columns.*
2. *Values stored in a column should be of the same domain*
3. *All the columns in a table should have unique names.*
4. *And the order in which data is stored, does not matter.*

Radhika Rani Chintala

1NF Example

Roll_no	Name	Subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

Three of the rules were satisfied

Roll_no	Name	Subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

Radhika Rani Chintala

2nd Normal Form (2NF)

For a table to be in the Second Normal Form:

1. *It should be in the First Normal form.*
2. *And, it should not have Partial Dependency.*

Radhika Rani Chintala

Partial Dependency ?

Dependency / Functional Dependency:

If the information stored in a table can uniquely determine another information in the same table, then it is called Functional Dependency.

P -> Q : *P functionally determines Q*

/ Q is functionally dependent on P

Partial Dependency:

Partial Dependency occurs when a nonprime attribute is functionally dependent on part of a candidate key.

Radhika Rani Chintala

2NF Example

Jump to Excel

Radhika Rani Chintala

2NF

- For a table to be in the Second Normal form, it should be in the First Normal form and it should not have Partial Dependency.
- Partial Dependency exists, when for a composite primary key, any attribute in the table depends only on a part of the primary key and not on the complete primary key.
- To remove Partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.

3rd Normal Form (3NF)

A table is said to be in the Third Normal form when,

1. *It is in the Second Normal form.*
2. *And, it doesn't have Transitive Dependency.*

Radhika Rani Chintala

Transitive Dependency ?

When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key, then we say it is ***Transitive Dependency***

How to remove Transitive Dependency?

- Again the solution is very simple.
- Take out the columns that caused transitive dependency from the existing table and move them to some other table where it fits in well.

Radhika Rani Chintala

3NF Example

Let's go to Excel

Radhika Rani Chintala

Boyce-Codd Normal Form (BCNF)

For a table to be in BCNF, following conditions must be satisfied:

- *R must be in 3rd Normal Form*
- *And, for each functional dependency (X → Y), X should be a super Key.*

Radhika Rani Chintala

Super Key

Candidate Key: The minimal set of attribute which can uniquely identify a tuple is known as candidate key.

Ex : Student table: {**Sno**} or {**Mobile**} ; Course table: {**Sno, Course_No**}

Super Key: The set of attributes which can uniquely identify a tuple is known as Super Key. A candidate key is a super key but vice versa is not true.

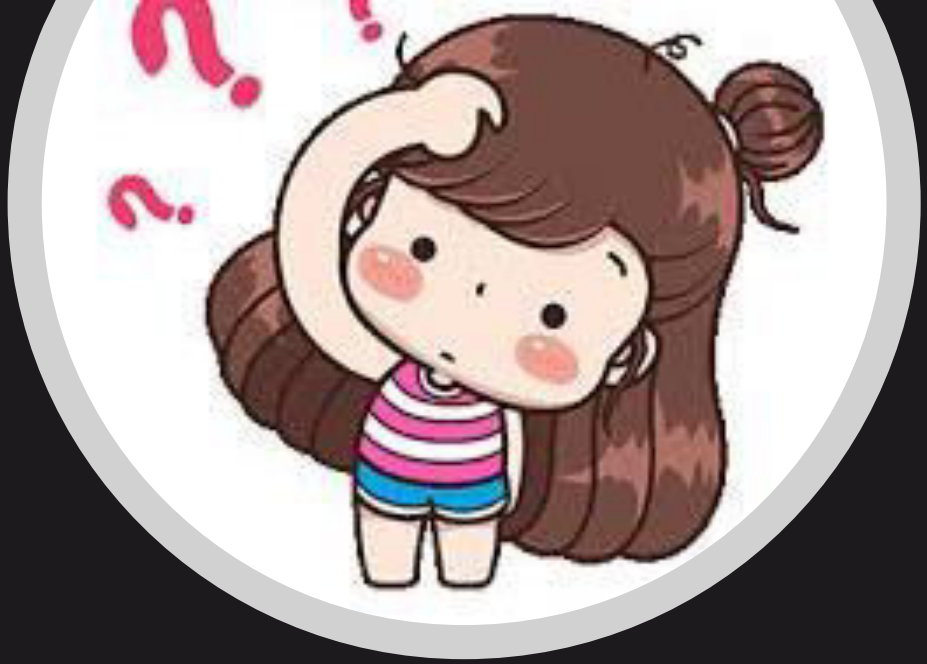
Ex : Student table: **{Sno}, {Sno, Sname}**

Primary Key: There can be more than one candidate key in relation out of which one can be chosen as the primary key.

Ex : Student table: **{Sno}**

Sno	SName	Mobile	Age	Address

Sno	Course_No	Course_Name



BCNF Example

Let's move to Excel

Radhika Rani Chintala

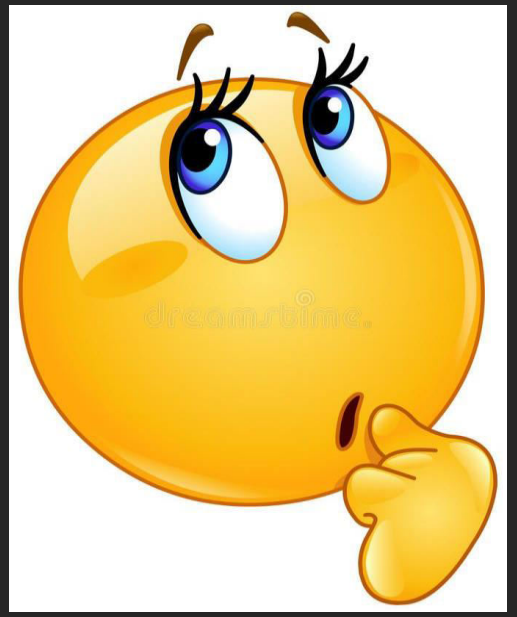
4th Normal Form (4NF)

A table is said to be in the Fourth Normal Form when,

1. *It is in the Boyce-Codd Normal Form.*
2. *And, it doesn't have Multi-Valued Dependency.*

Radhika Rani Chintala

Multi-Valued Dependency



?

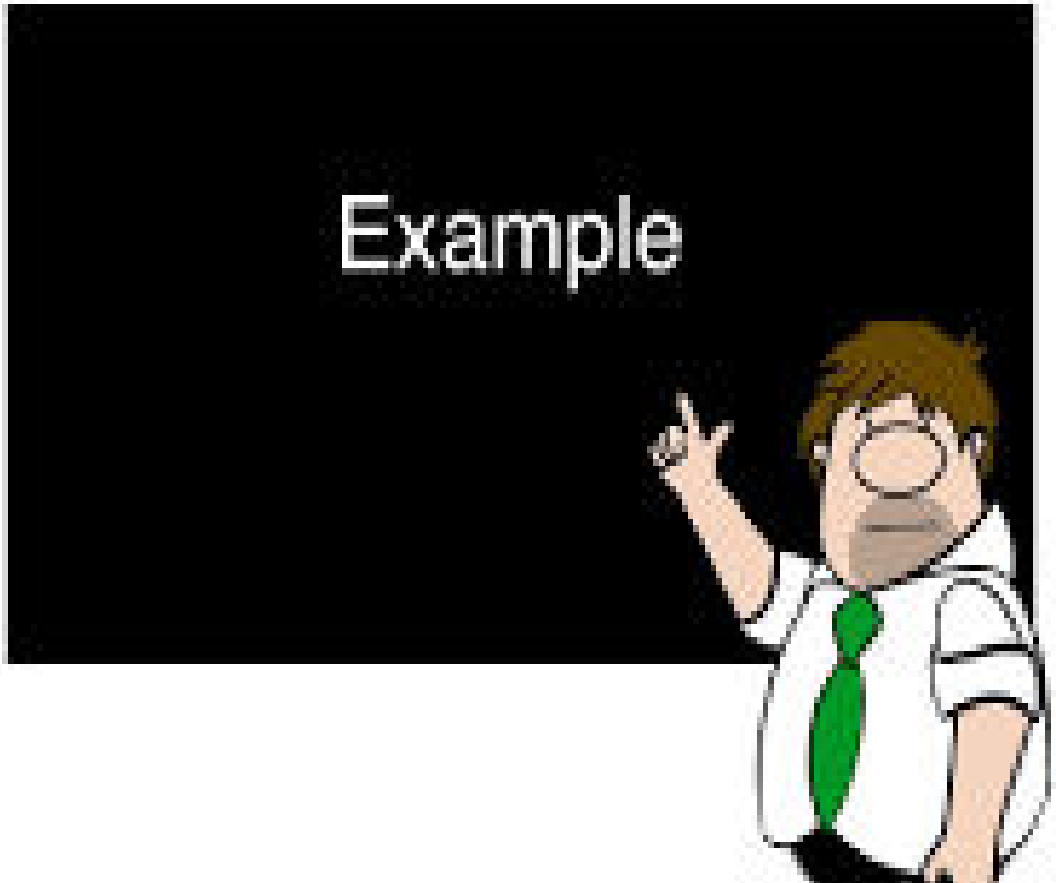
A table is said to have multi-valued dependency, if the following conditions are true.

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.

2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.

3. And, for a relation `R(A,B,C)`, if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation (table), it is said to have multi-valued dependency.



4NF Example

Move to Excel

Radhika Rani Chintala

5th Normal Form (5NF)

A table is said to be in the Fifth Normal Form when,

1. *It is in 4NF*
2. *And, it doesn't have any Join Dependency.*

- **5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.**
- **Once it is in fifth normal form it cannot be broken into smaller relations without changing the facts or the meaning.**

Radhika Rani Chintala

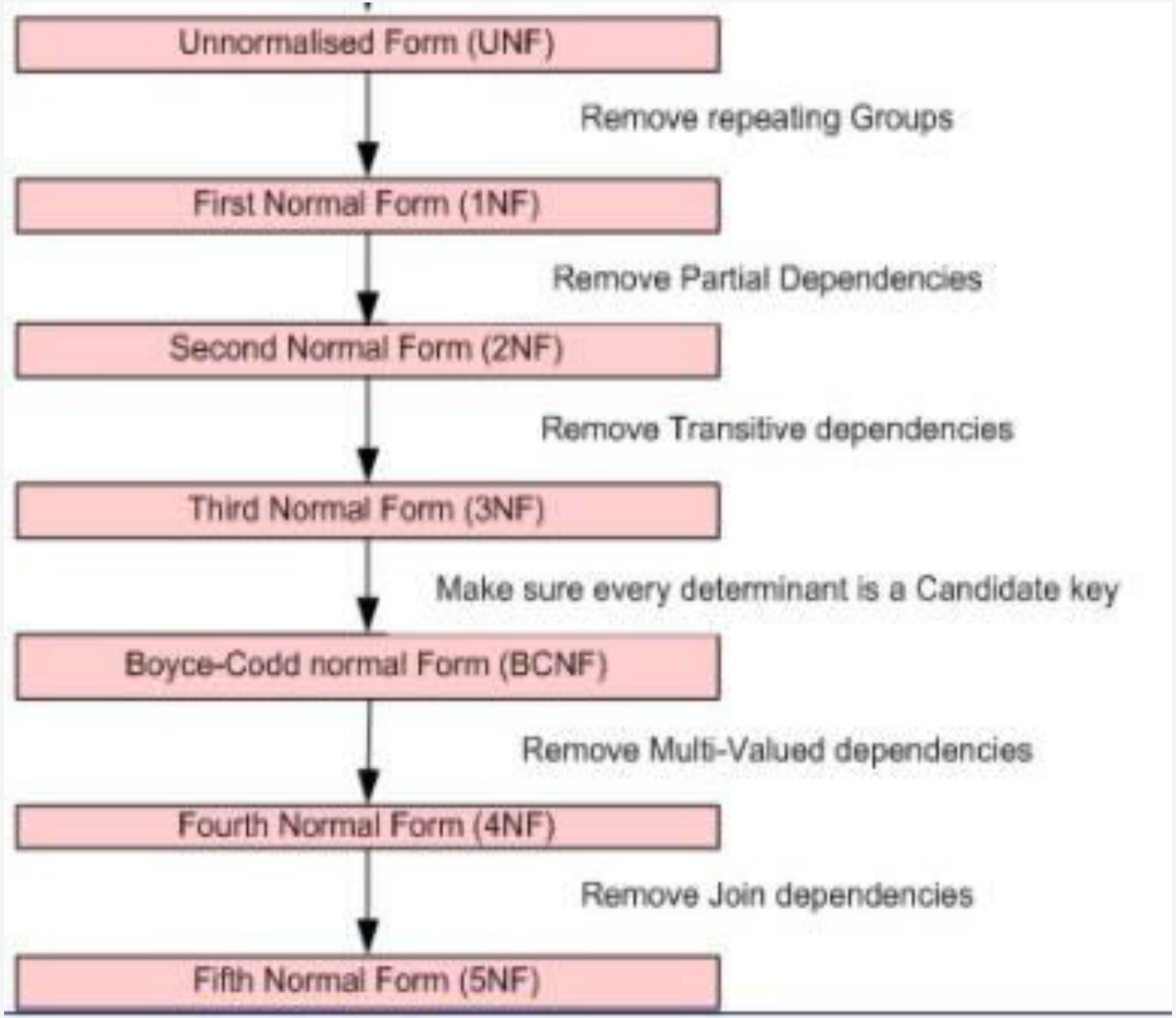
Join Dependency

?

- If a table can be recreated by joining multiple tables and each of this table have a subset of the attributes of the table, then the table is in **Join Dependency**.
- 5NF is also known as Project-join normal form (PJNF).
- Fifth Normal Form in Database Normalization is generally not implemented in real life database design.

Radhika Rani Chintala

Summary



Radhika Rani Chintala

THE END

Radhika Rani Chintala

Before Normalization

Roll_no	Name	Subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

After Normalization

Roll_no	Name	Subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

Before Normalization

STUDENT				
sid	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat
12	Bkon	09-WY	IT	Rajasthan

SU
subject_id
1
2
3

sid -> name
sid -> branch

SCORE			
sid	sub_id	marks	teacher
10	1	70	Java Teacher
10	2	75	C++ Teacher
11	1	80	Java Teacher

Candidate Key: { sid, sub_id }

teacher is a non-prime attribute that is dependent only on sub_id which is part of candidate key hence there is partial dependency and we need to remove it.

After Normalization

BJECT

subject_name
Java
C++
Php

STUDENT

sid	name	reg_no
10	Akon	07-WY
11	Akon	08-WY
12	Bkon	09-WY

SUBJECT

sub_id	sub_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

branch	address
CSE	Kerala
IT	Gujarat
IT	Rajasthan

SCORE		
sid	sub_id	marks
10	1	70
10	2	75
11	1	80

Before Normalization

STUDENT				
sid	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat
12	Bkon	09-WY	IT	Rajasthan

SUBJECT		
sub_id	sub_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

SCO		
sid	sub_id	marks
10	1	70
10	2	75
11	1	80

In the Score table, we need to store some more information, which is the exam name and max
exam_name depends on sub_id because if you take Workshop lab it will be only for ME studen
Similarly exam_name depends on sub_id also. Because few subjects will have practical compor
Hence exam_name fully depends on primary key {sid,sub_id}

Max_marks just depends on exam_name. if exam is workshop, then max marks is 200. if exam
Thus, Max_marks(non-prime attribute) depends only on exam_name(non-prime) which depen
Hence there is a transitive dependency and we need to remove it to form 3NF.

How to remove Transitive Dependency?

Again the solution is very simple. Take out the columns `exam_name` and `max_ma`

After Norn

STUDENT

sid	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat
12	Bkon	09-WY	IT	Rajasthan

)RE

exam_name	Max_marks

SCORE

sid	sub_id	marks	exam_id
10	1	70	2
10	2	75	2
11	1	80	1

:imum marks, so let's add 2 more columns to the Score table.
ıts, not for CSE students.
ıent and few doesn't.

is external, then max marks is 100 and so on.
ds on Primary key.

`rks` from Score table and put them in an **Exam** table and use the `exam_id` wherever re

nalization

SUBJECT		
sub_id	sub_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

EXAM		
exam_id	exam_name	Max_marks
1	Workshop	200
2	External	100
3	Practicals	50

equired.

Before Normalization

STUDENT		
sid	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

In the table above:
One student can enrol for multiple subjects. For example, student with **student_id** 101, ha
For each subject, a professor is assigned to the student.
And, there can be multiple professors teaching one subject like we have for Java.
What do you think should be the **Primary Key**?
Well, in the table above `student_id, subject` together form the primary key, because u
One more important point to note here is, one professor teaches only one subject, but one
Hence, there is a dependency between `subject` and `professor` here, where `subject` de
This table satisfies the **1st Normal form** because all the values are atomic, column name
This table also satisfies the **2nd Normal Form** as their is no **Partial Dependency**.
And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal F**
But this table is not in **Boyce-Codd Normal Form**.

Why this table is not in BCNF?

In the table above, `student_id, subject` form primary key, which means `subject` colu
But, there is one more dependency, `professor` → `subject`.
And while `subject` is a prime attribute, `professor` is a **non-prime attribute**, which is not

How to satisfy BCNF?

To make this relation(table) satisfy BCNF, we will decompose this table into two tables, **st**

After Normalization

STUDENT	
sid	pid
101	1
101	2
102	3
103	4
104	1

PROFESSOR	
pid	professor
1	P.Java
2	P.Cpp
3	P.Java2
4	P.Chash

s opted for subjects - Java & C++

sing `student_id` and `subject`, we can find all the columns of the table.
subject may have two different professors.
pends on the professor name.
are unique and all the values stored in a particular column are of same domain.

orm.

ın is a **prime attribute**.

allowed by BCNF.

ident table and **professor** table.

subject
Java
C++
Java
C#

Before Normalization

COLLEGE_ENROLLMENT TABLE		
s_id	**course**	**hobby**
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

To make the a

CourseOp
s_id
1
1
2
2

As you can see in the table above, student with `s_id` **1** has opted for two courses,
You must be thinking what problem this can lead to, right?
Well the two records for student with `s_id` **1**, will give rise to two more records, as

COLLEGE_ENROLLMENT TABLE		
s_id	**course**	**hobby**
1	Science	Cricket
1	Science	Hockey
1	Maths	Cricket
1	Maths	Hockey

And, in the table above, there is no relationship between the columns `course` and
So there is multi-value dependency, which leads to un-necessary repetition of data

After Normalization

above relation satify the 4th normal form, we can decompose the table into 2 tables.

ted Table

course
Science
Maths
C#
Php

Hobbies Table

s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Science and **Maths**, and has two hobbies, **Cricket** and **Hockey**.

shown below, because for one student, two hobbies exists, hence along with both the cour

f 45thn

hobby. They are independent of each other.

and other anomalies as well.

ses, these hobbies should be specified.

File Organization

Radhika Rani Chintala

File Organization

Part-1

Agenda

- Introduction
- Storage Devices
- Secondary Storage devices
- Hardware Mechanism of Disk drive
- Tertiary Storage devices

Introduction

The collection of data that makes up a computerized database must be stored physically on some computer **storage medium**.

The DBMS software can then retrieve, update, and process this data as needed.

Computer storage media form a **Storage Hierarchy** that includes two main categories:

- **Primary storage**
- **Secondary storage**
- **Tertiary storage**

Storage Devices

Primary Storage

- Direct access by CPU.
- Fast access to data
- Limited storage capacity.
- More expensive.
- Volatile
- Main memory & faster cache memory.

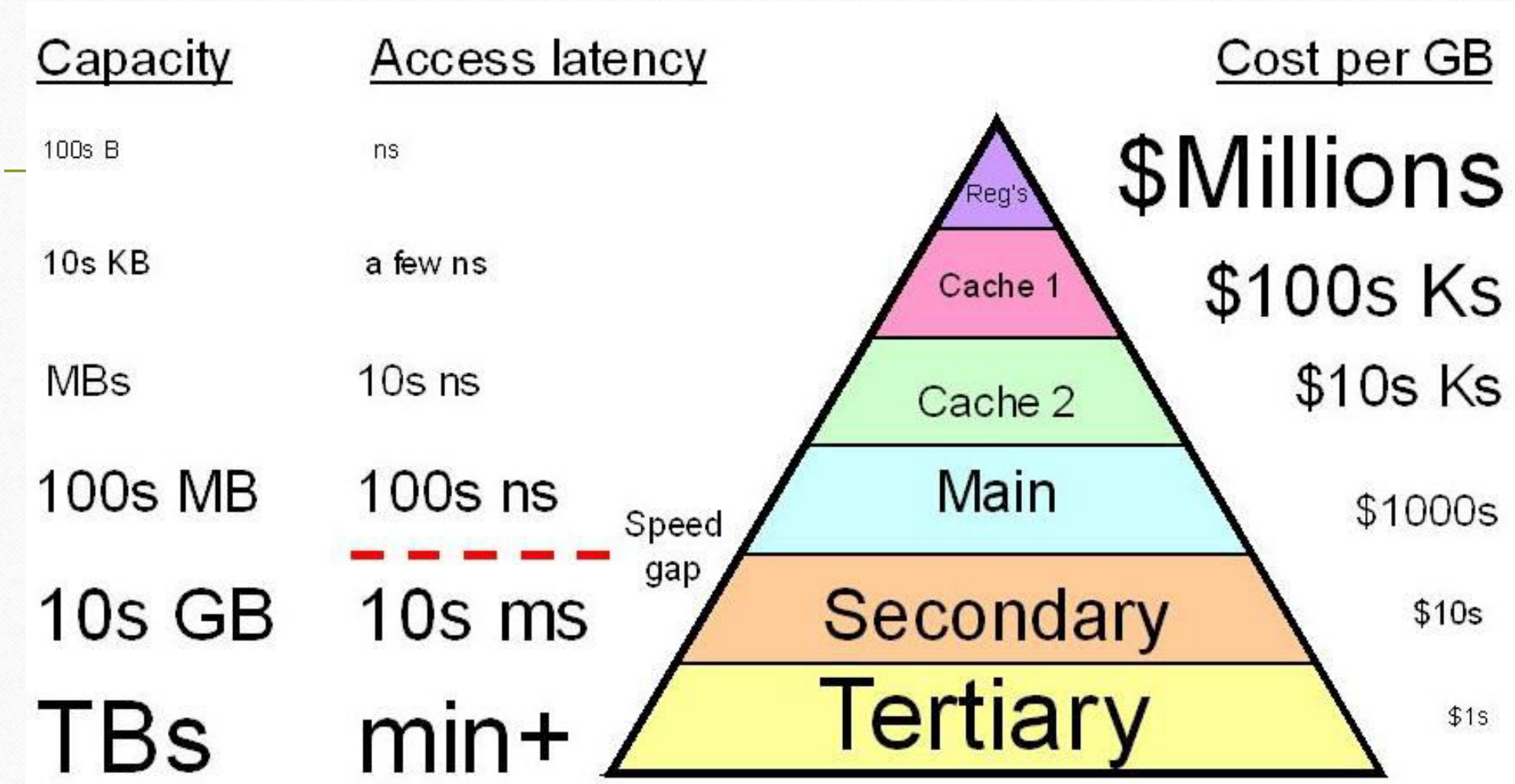
Secondary Storage

- No direct access to CPU
- Slow access to data
- Large capacity
- Less expensive
- Non-Volatile
- Online devices
- Magnetic disk & Flash memory
- Most databases are stored permanently on magnetic disks

Tertiary Storage

- No direct access to CPU
- Slower access to data
- Large capacity
- Less expensive than disks
- Non-Volatile
- Offline devices
- Magnetic tapes & Optical disks
- Used for backing up databases

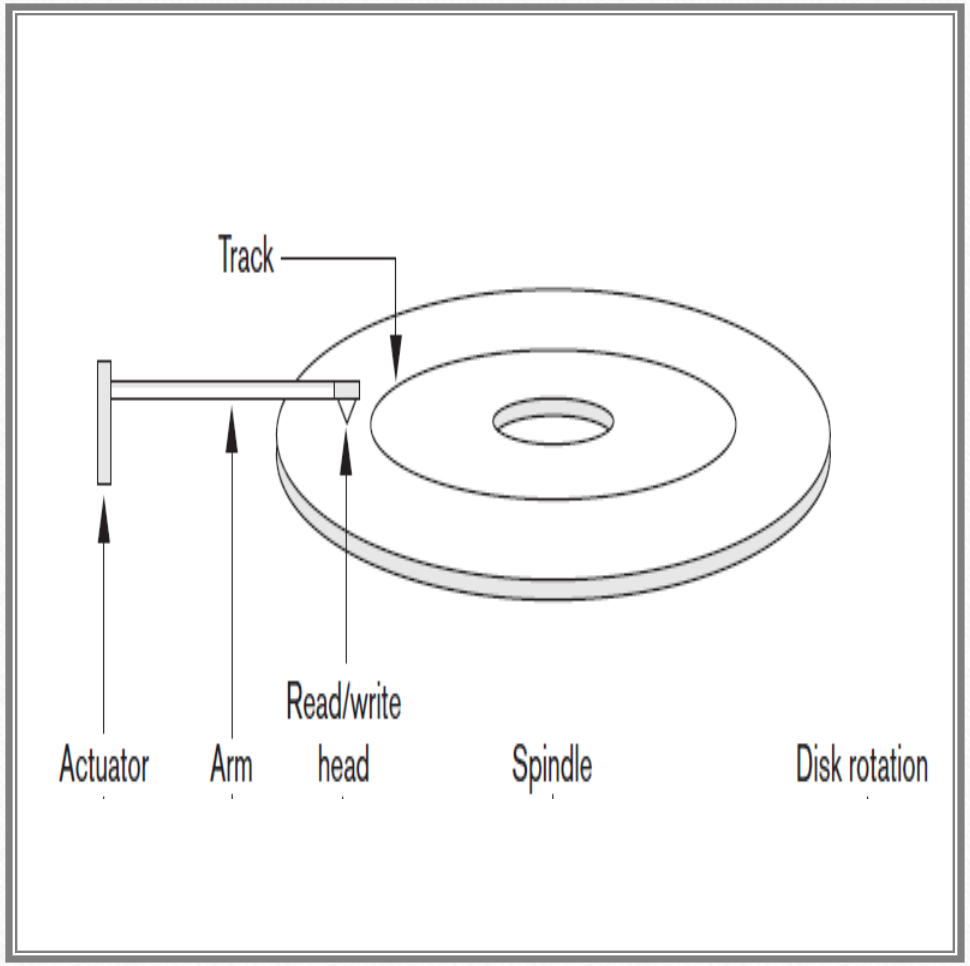
Storage Hierarchy



Secondary Storage Devices

Magnetic Disk

- Stores large amounts of data.
- The device that holds the disks is referred to as a **hard disk drive**, or **HDD**.
- Basic unit of data is a **bit.**
- By magnetizing an area on a disk, it can represent a bit value of either 0 or 1.
- Bits are grouped into **bytes** (or **characters**) -- > 4 to 8 bits.
- **Capacity** of a disk = No. of bytes it can store
- Hard disks can hold from several hundred gigabytes up to a few terabytes.



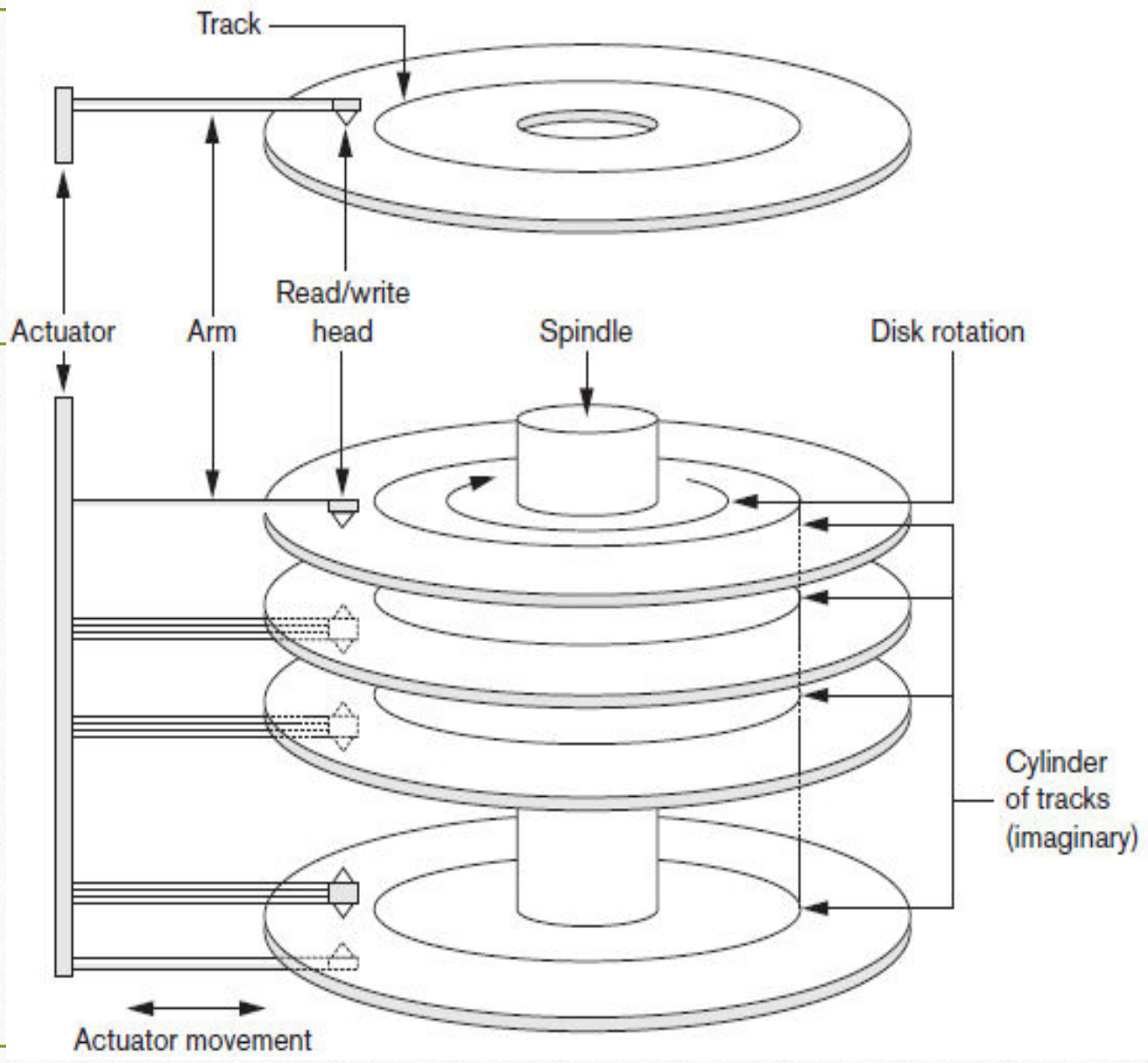
Hardware description of Disk Devices

- Made of magnetic material
- Shaped as a thin circular disk
- Protected by a plastic or acrylic cover.
- Single-sided disk and Double-sided disk.
- 3.5” & 2.5” diameter.
- Disks are assembled into a **disk pack,** which may include many disks and therefore many surfaces**.**

- **Disk Pack:** Which include multiple disks and thus many surfaces.

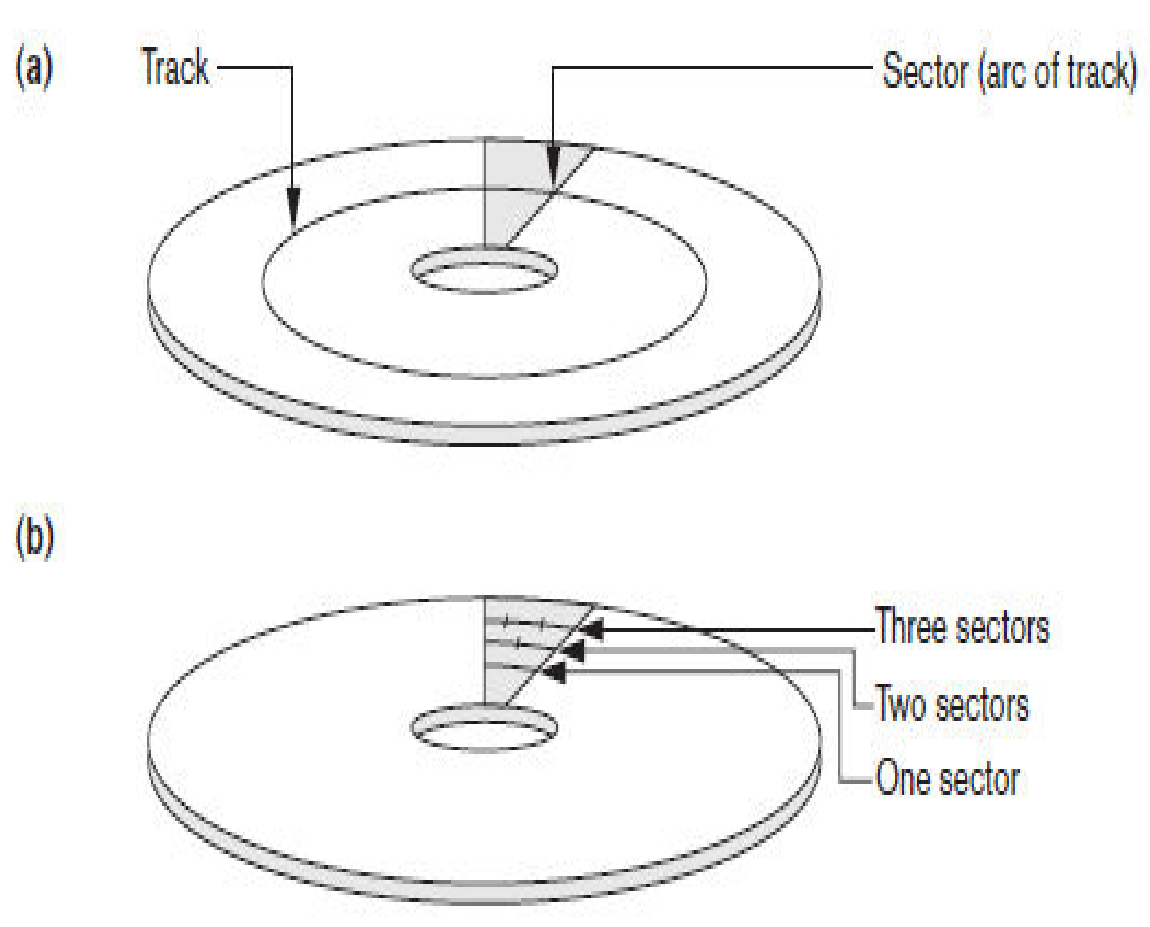
- **Tracks:** Information is stored on a disk surface in concentric circles of small width, where each circle is called a **track**.

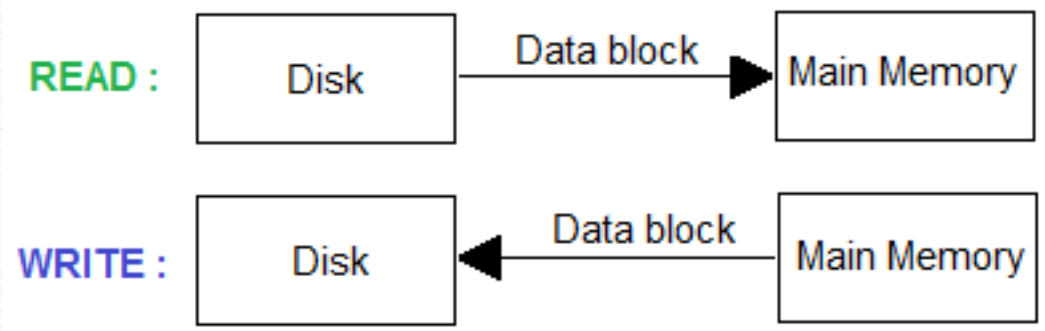
- **Cylinder:** Tracks with the same diameter on the various surfaces are called a **cylinder.**



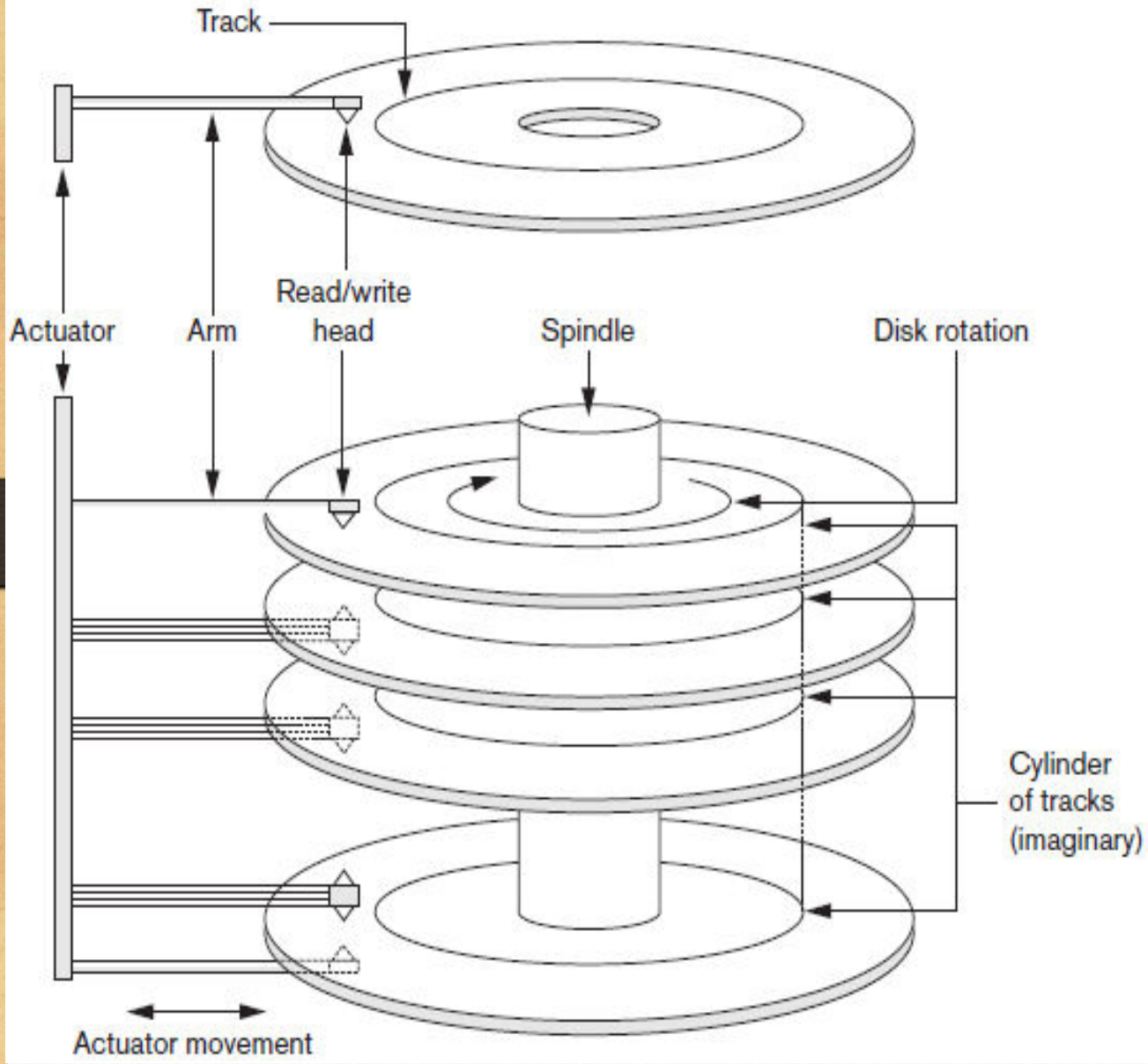
- **Sectors:** Track usually contains a large amount of information, so it is divided into smaller blocks called sectors.

- **Disk Block:** The division of a track into equal-sized **disk blocks** (or **pages**) is set by the operating system during disk formatting (or initialization).
- Sectors subdivided or combined into blocks during initialization.





- **Data Transfer:** Data transfer b/w main memory and disk takes place in units of disk blocks.
- **Hardware Block Address** = (cylinder number, track number, block number)
- **Logical block address(LBA)** = number between 0 and n (assuming the total capacity of the disk is n + 1 blocks).
- **Buffer:** a contiguous reserved area in main memory that holds one disk block.
- **Cluster**(several contiguous blocks) may be transferred as a unit. In this case, the buffer size is adjusted to match the number of bytes in the cluster.



Hardware Mechanism of Disk drive

- Disk Drive
- Read/Write head
- Mechanical Arm
- Actuator
- Disk Types
  - Fixed-head disks
  - Movable-head disks

Interfacing Disk Drives to Computer Systems

Disk Controller : Controls the disk drive and interfaces it to the computer system.

Seek time	• Time taken by the disk controller to position the read/write head on the correct track.
Rotational delay or latency	• Rotational time taken to position the beginning of the desired block under the read/write head.
Block transfer time	• Time taken to transfer the data
Bulk Transfer	• Transfer several consecutive blocks on the same track or cylinder.

Total Block Transfer Time Calculation

Total time = Seek time + Rotational delay + Block transfer time

*Bulk transfer rate = Seek time + Rotational delay + (n * Block transfer time)*

Tertiary Storage Devices

Magnetic Tapes

- Sequential access devices .
- high-capacity.
- Data is stored on reels. Similar to audiotapes or videotapes.
- A tape drive is required
- A read/write head is used to read or write data blocks on tape.
- Blocks may be larger than those for disks
- Slow access devices
- Used for Backing up the database

- Storage Devices
- Magnetic Disks
- Hardware Mechanism of Disk drive
- Magnetic Tapes

Summary of Part-1

File Organization

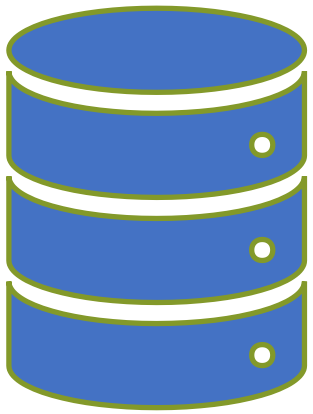
Part-2

File Organization

Part-2

Agenda

- Buffer Management
- Various ways of formatting and storing file records on disk.
- Various types of operations that are typically applied to file records.
- Three primary methods for organizing file records on disk:
  - unordered records
  - ordered records
  - hashed records.



Buffer Management

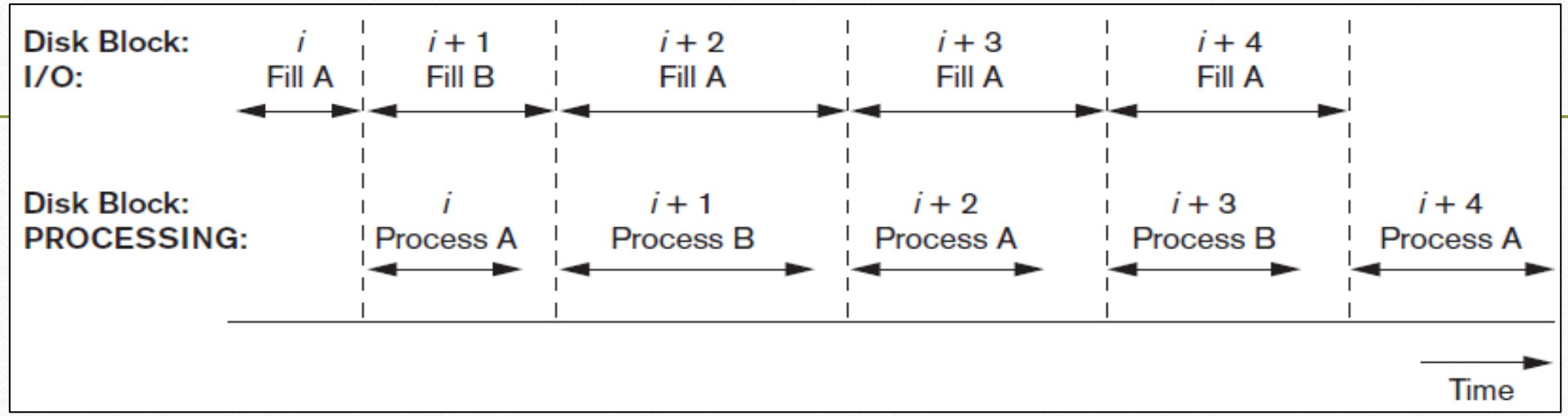
Buffering of Blocks

Multiple Buffers: When several blocks need to be transferred from disk to main memory, several buffers can be reserved in main memory to speed up the transfer.

Disk I/O Processor: Controller that takes care of I/O operations.

Parallel processing of Buffers: While one buffer is being read or written, CPU can process data in the other buffer. CPU processing and Disk I/O processing can be done parallelly.

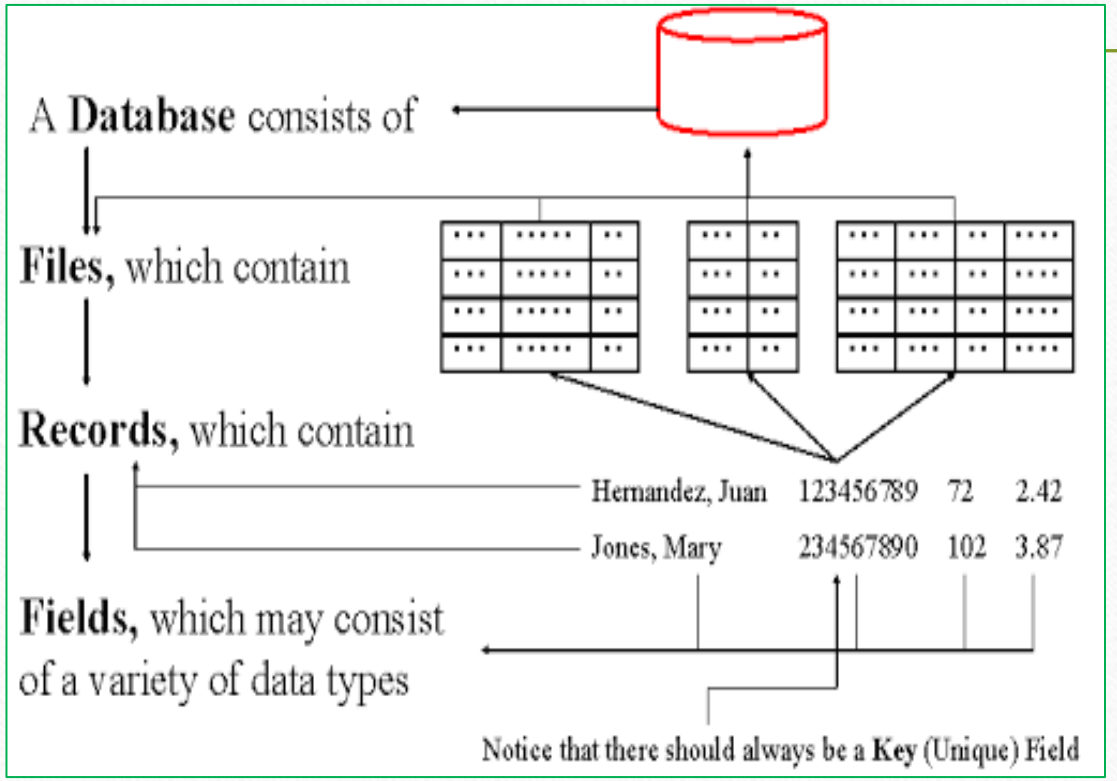
Double Buffering



- Reading and processing can proceed in parallel.
- The CPU can start processing a block once its transfer to main memory is completed; at the same time, the disk I/O processor can be reading and transferring the next block into a different buffer.

Placing File Records on Disk

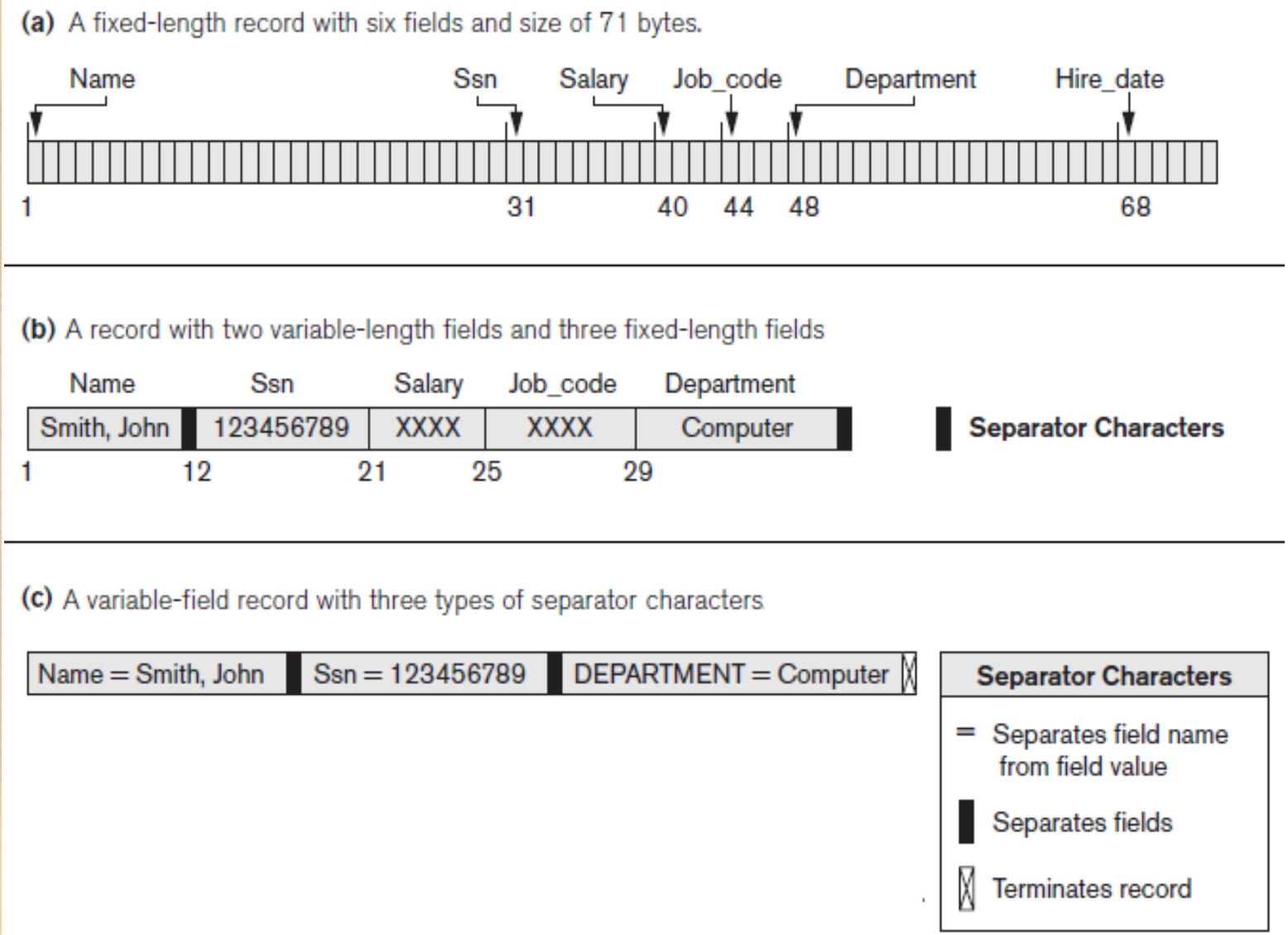
Files, Fixed-Length Records, and Variable-Length Records



- **File:** a sequence of records. All records in a file are of the same record type.
- **Fixed-length records:** Every record in the file has exactly the same size (in bytes).
- **Variable-length records:** Different records in the file have different sizes.

Three Storage Formats

- Fixed Length Record
- Variable Length Record with Separator
- Variable Length Record with multiple Separators

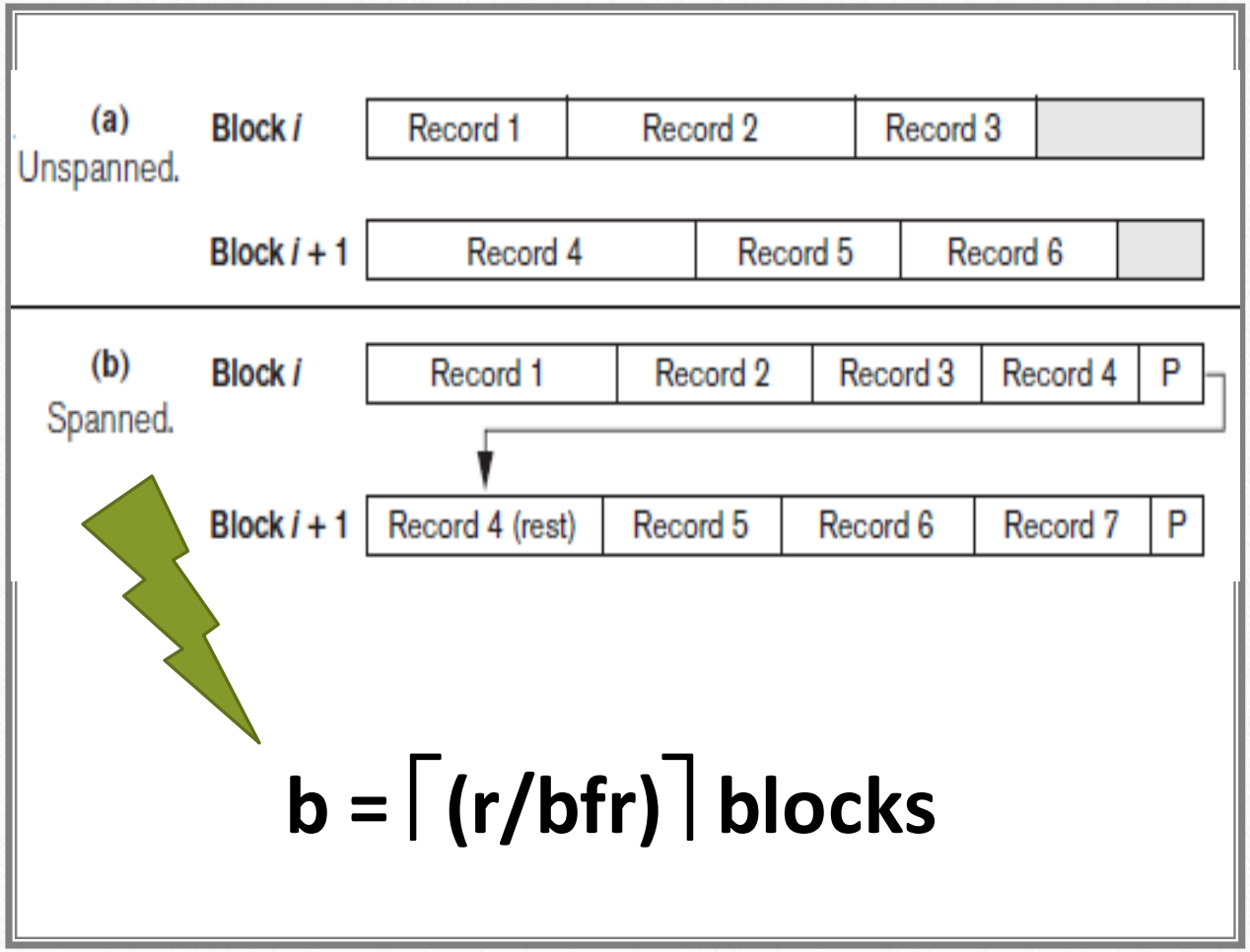


Record Blocking

- When the **block size > record size**, each block will contain numerous records
- When the **record size > block size**, each record occupies multiple blocks.
- For a file of fixed-length records of size *R* bytes, with *B* >= *R*, we can fit

bfr (blocking factor) = $\lfloor B/R \rfloor$ records per block

Unused space in each block = B – (bfr * R) bytes



$$b = \lceil (r/bfr) \rceil \text{ blocks}$$

Spanned & Unspanned Records

- **Unspanned Records:**
  - A record is found in one and only one block.
  - Records do not span across block boundaries.
  - Used with fixed-length records having $B > R$
- **Spanned Records:**
  - Records are allowed to span across block boundaries.
  - Used with variable-length records having $R > B$

Allocating File blocks on Disk

Contiguous Allocation

- The file blocks are allocated to consecutive disk blocks.
- Reading a file is very fast.
- But expanding a file is difficult.

Linked Allocation

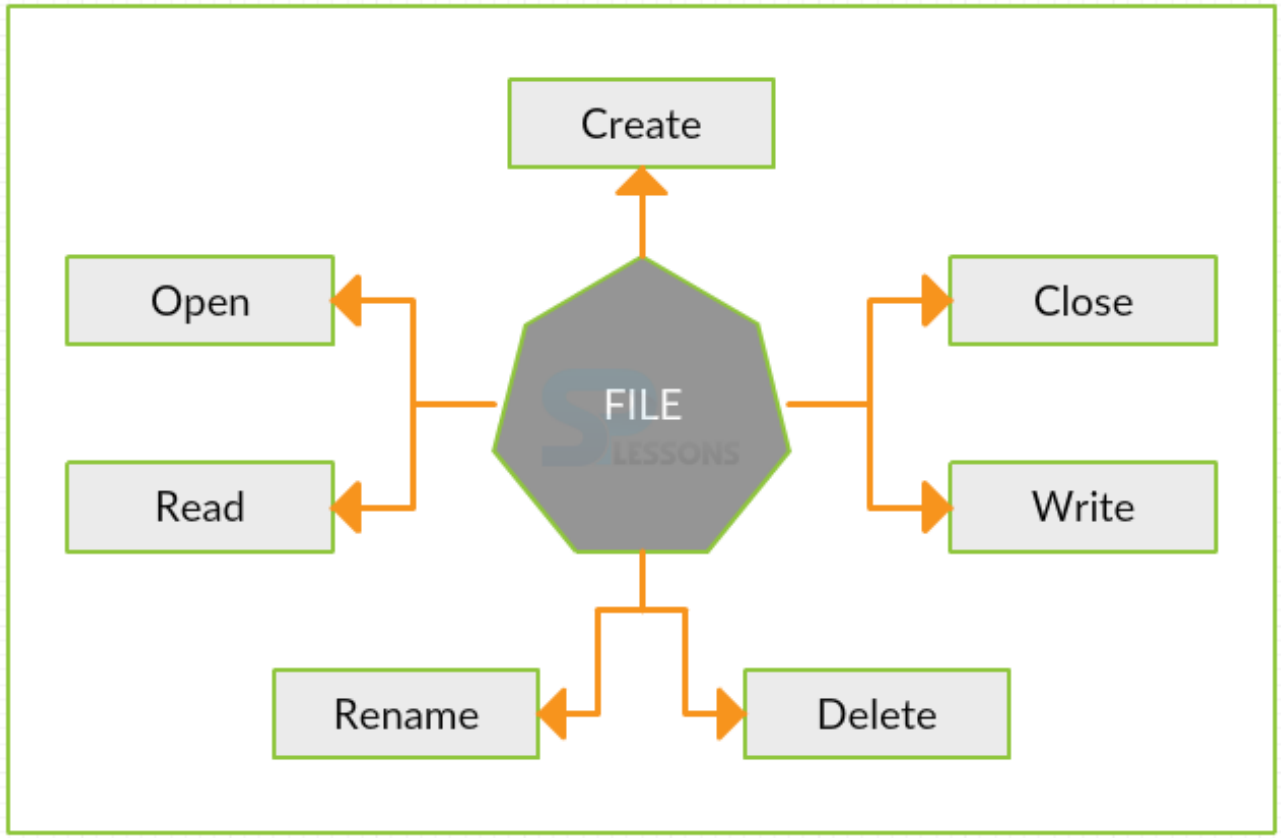
- Each file block contains a pointer to the next file block.
- Easy to expand a file
- But reading a file is slow

Indexed Allocation

- One or more **index blocks** contain pointers to the actual file blocks.

It is also common to use combinations of these techniques.

Operations on Files



File Operations

- Open
- Reset
- Find (or Locate)
- Read (or Get)
- FindNext
- Delete
- Modify
- Insert
- Close
- Scan
- FindAll
- Find (or Locate) n
- FindOrdered
- Reorganize

Files of Unordered Records

and

Ordered Records

Files of Unordered Records and Ordered Records

Files of Unordered Records

- Also called as **Heap Files or Pile Files.**
- Records are stored in the same order in which they are created.
- **Insert operation:** *Fast*
- **Search (or Update) operation:** *Slow*
- **Delete operation:** *Slow*
- Deleting a record creates a hole in the page.

Files of Ordered Records

- Also called as **Sorted Files.**
- **Ordering Field**: Records are sorted on the values of one or more fields.
- **Insert operation:** *Poor*
- **Search (or Update) operation:** *Fast*
- **Delete operation:** *Fast*

Summary of Part-2

- Buffer Management
- Placing File Records on Disk
- Operations on Files
- Files of Unordered and Ordered Records

DATABASE INDEXING

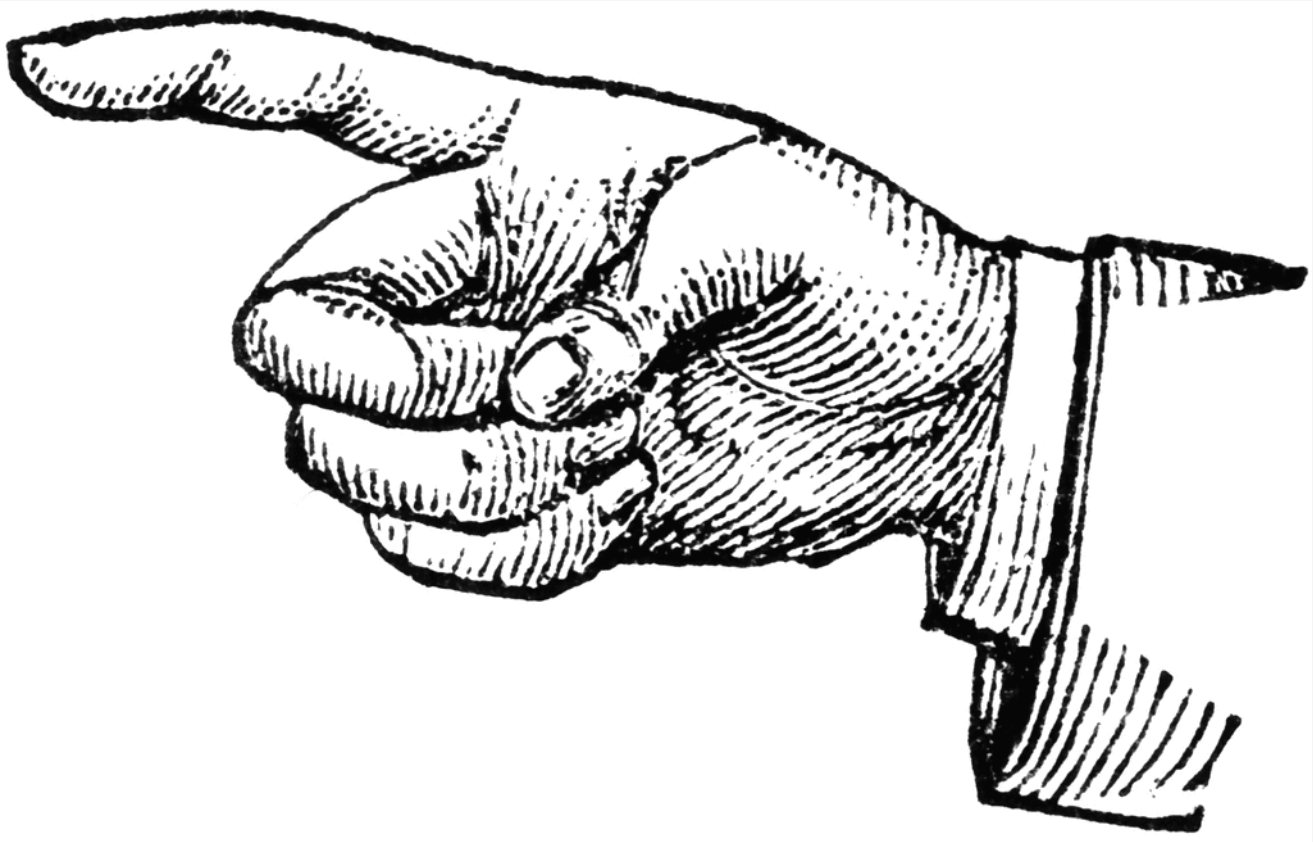
Radhika Rani Chintala

Session Contents

- What is Index / Indexing?
- Why we need Index / Indexing?
- What is Index in Databases?
- Different types of Indexing
- Importance of Indexing

Radhika Rani Chintala

What is Indexing ?

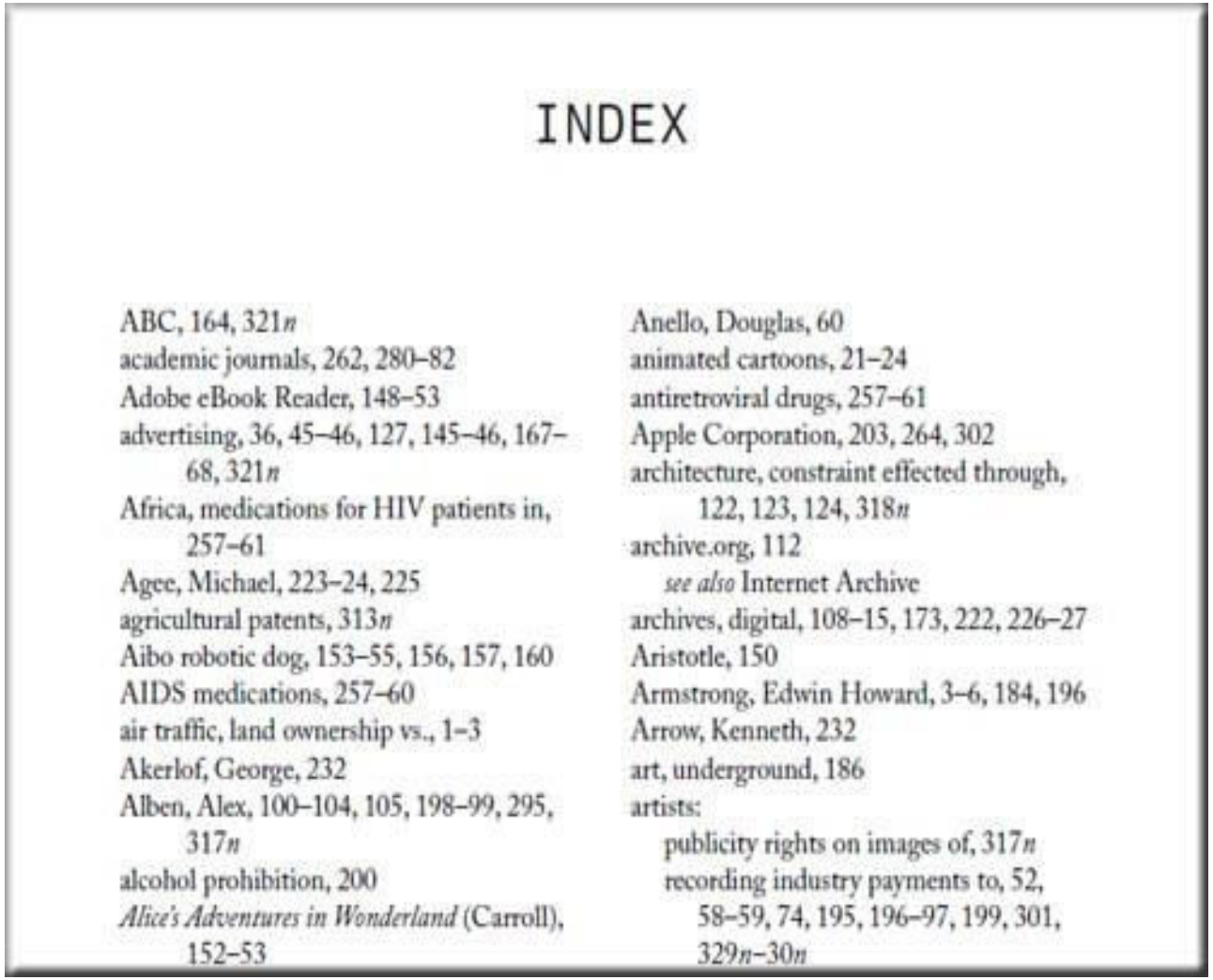


INDEXING is a data structure technique which allows you to quickly retrieve records from a database file.

WHY INDEXING IS USED?

Radhika Rani Chintala

?

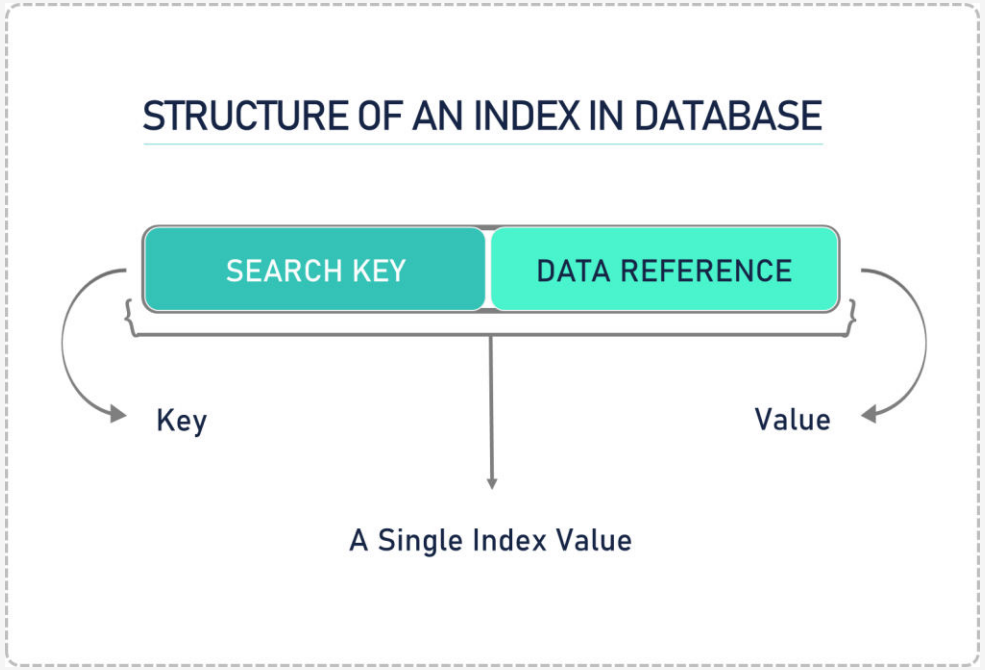


INDEX

ABC, 164, 321*n*
academic journals, 262, 280–82
Adobe eBook Reader, 148–53
advertising, 36, 45–46, 127, 145–46, 167–68, 321*n*
Africa, medications for HIV patients in, 257–61
Agee, Michael, 223–24, 225
agricultural patents, 313*n*
Aibo robotic dog, 153–55, 156, 157, 160
AIDS medications, 257–60
air traffic, land ownership vs., 1–3
Akerlof, George, 232
Alben, Alex, 100–104, 105, 198–99, 295, 317*n*
alcohol prohibition, 200
Alice's Adventures in Wonderland (Carroll), 152–53
Anello, Douglas, 60
animated cartoons, 21–24
antiretroviral drugs, 257–61
Apple Corporation, 203, 264, 302
architecture, constraint effected through, 122, 123, 124, 318*n*
archive.org, 112
see also Internet Archive
archives, digital, 108–15, 173, 222, 226–27
Aristotle, 150
Armstrong, Edwin Howard, 3–6, 184, 196
Arrow, Kenneth, 232
art, underground, 186
artists:
publicity rights on images of, 317*n*
recording industry payments to, 52, 58–59, 74, 195, 196–97, 199, 301, 329*n*–30*n*

Indexes are used to quickly locate data without having to search every record in multiple disk blocks

Similar to Indexing in Textbooks



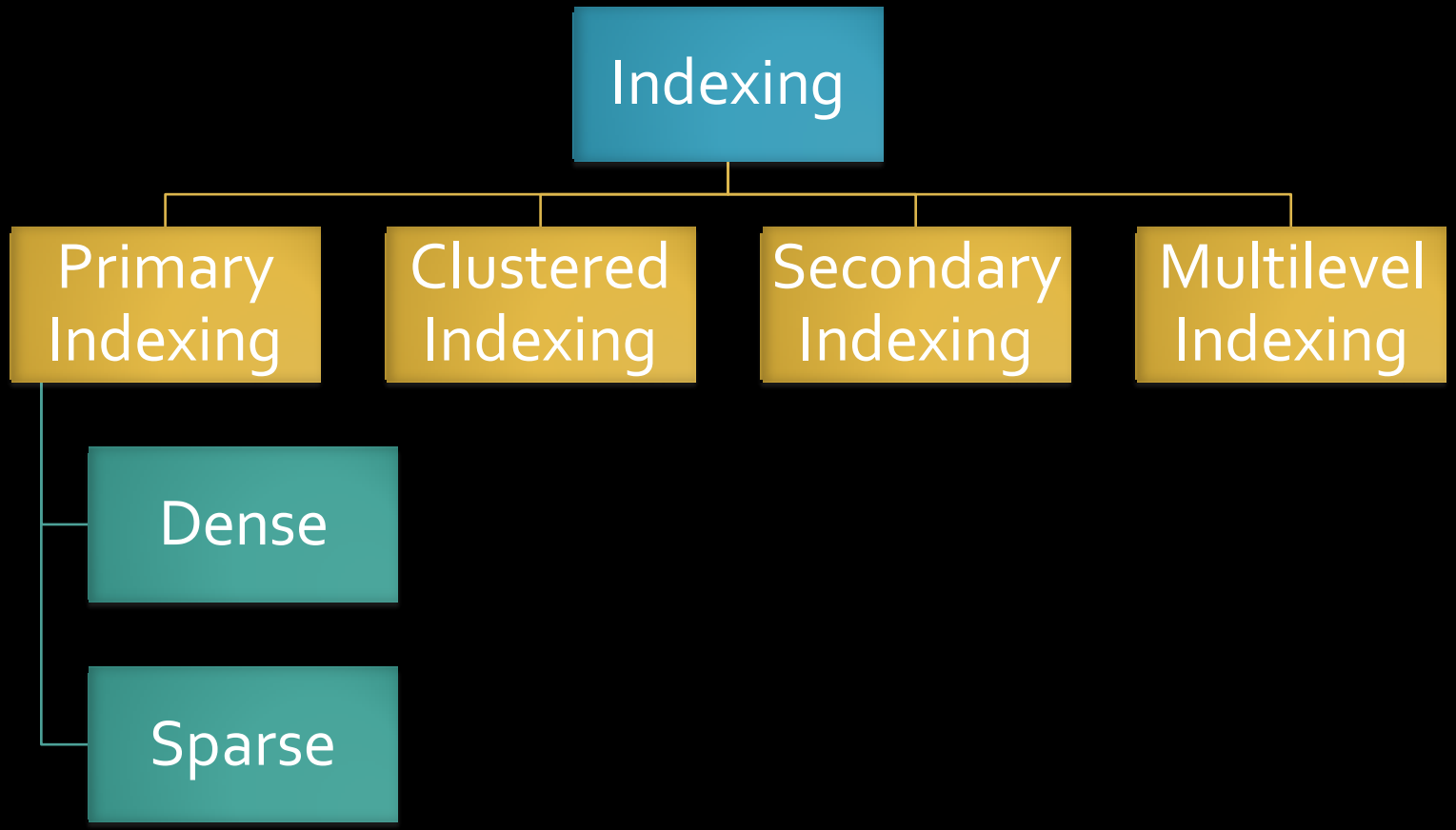
What is Index in Databases ?

- *An Index is a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.*

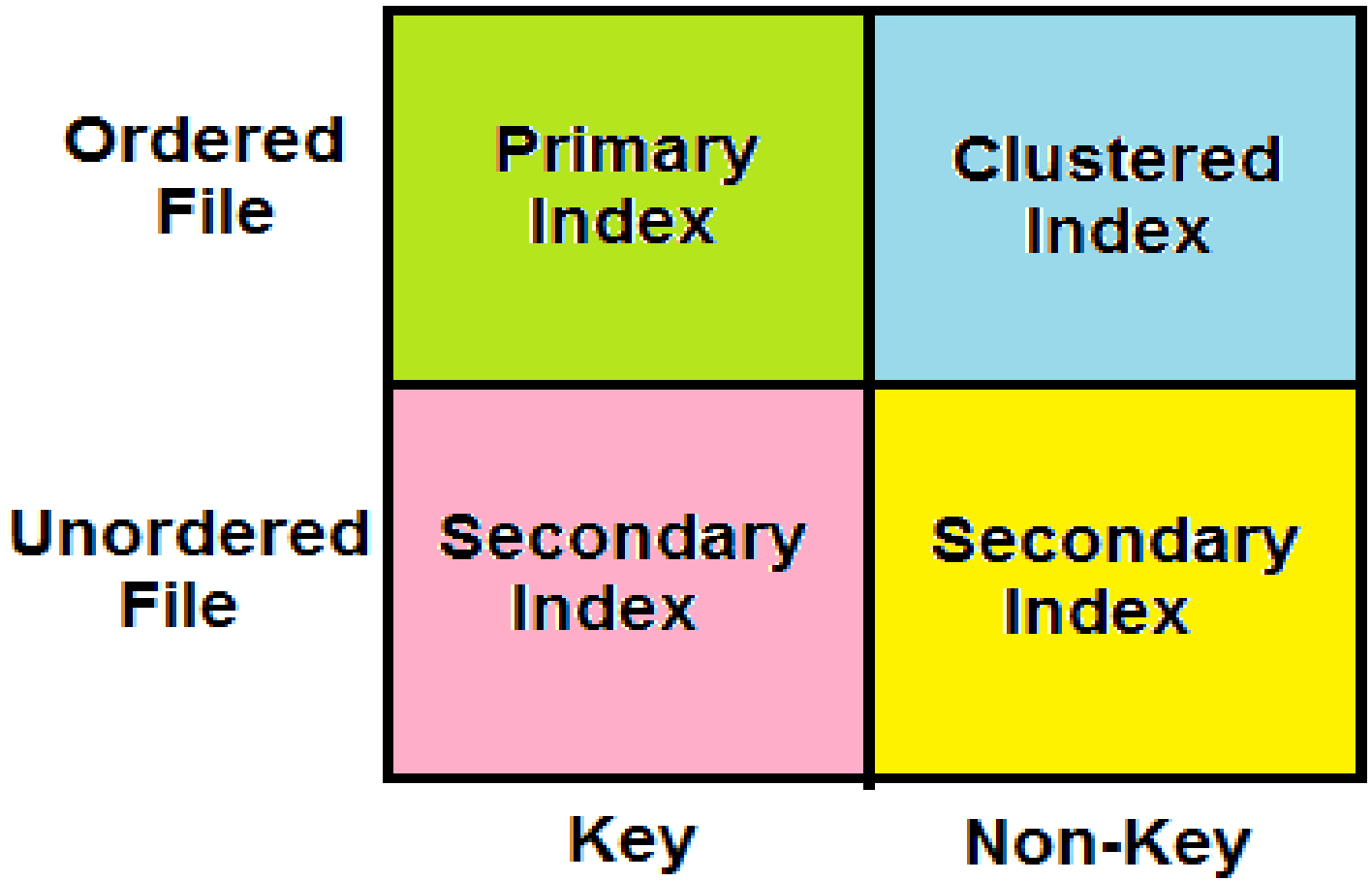
An index -

- **Takes a search key as input**
- **Efficiently returns a collection of matching records.**

Types of Indexing



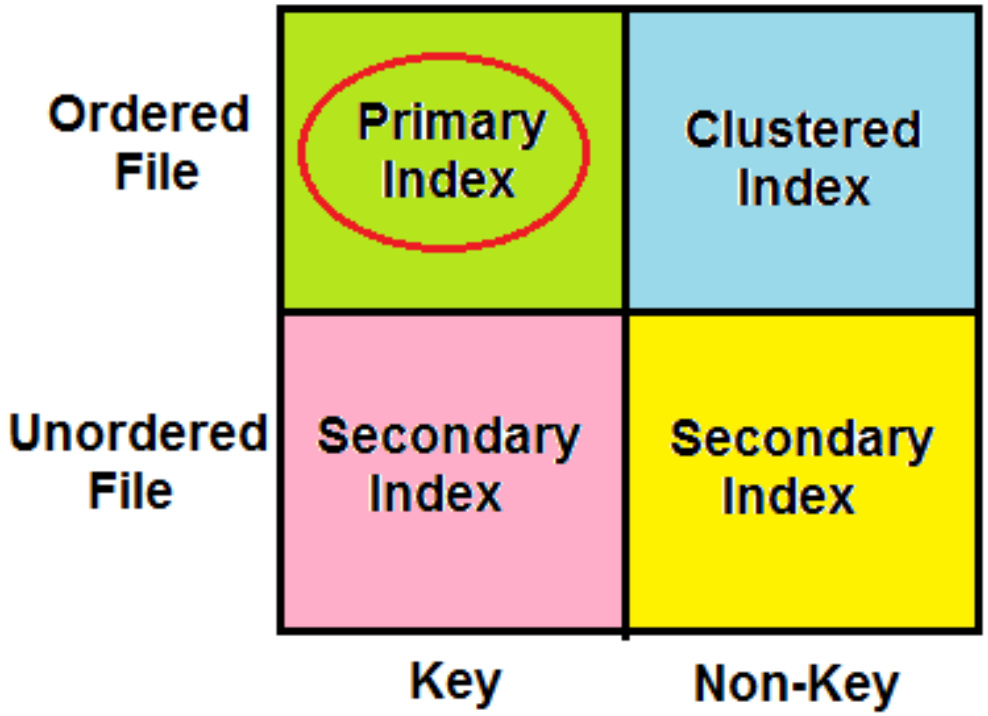
Radhika Rani Chintala



Which Indexing method is used ?

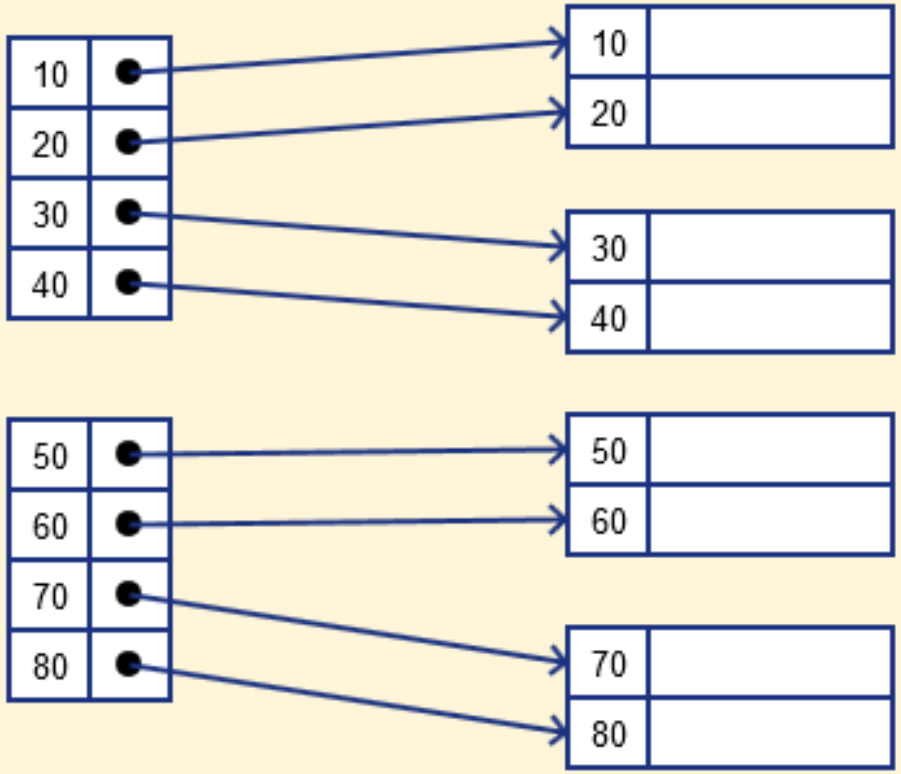
PRIMARY INDEXING

- Primary Index is an ordered file which is of fixed length size with two fields.
- The first field is the same a primary key and second field is a pointer that points to that specific data block.
- The primary Indexing is further divided into two types.
  - Dense Index
  - Sparse Index



Radhika Rani Chintala

Dense Index



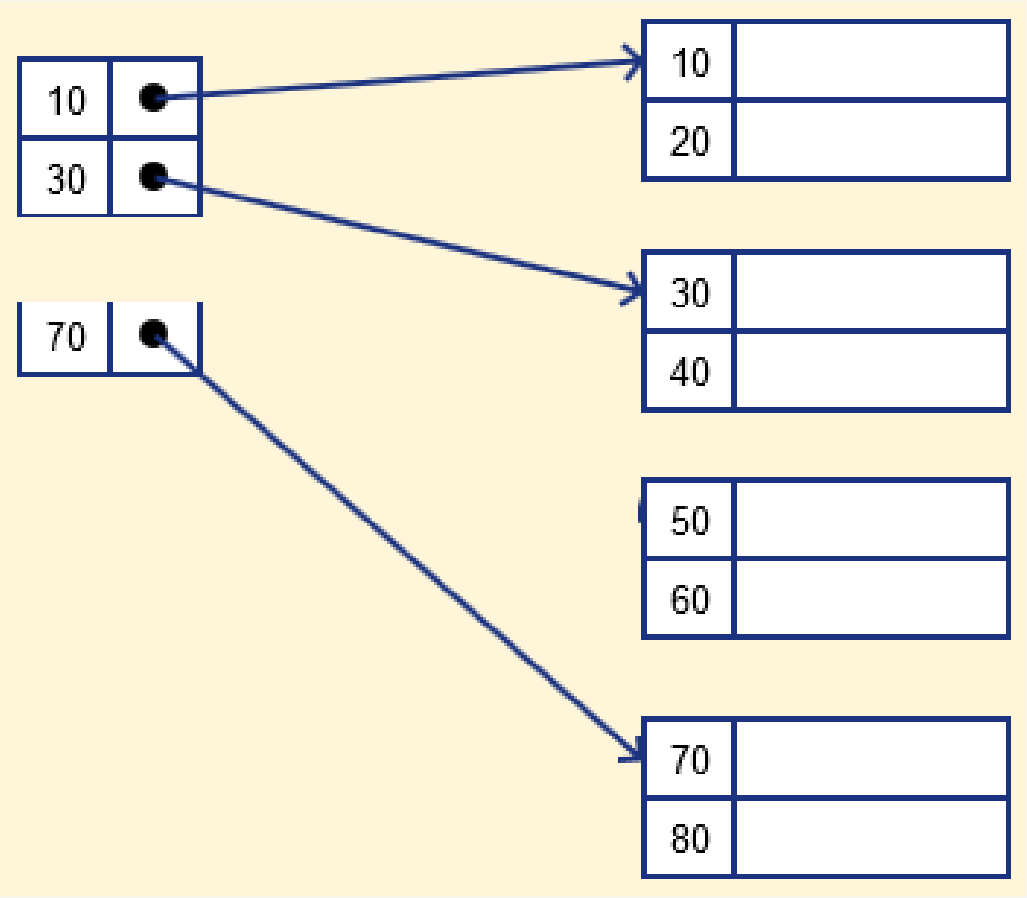
A record is created for each search key valued in the database.

Searching is faster

Requires more space to store index records

No. of records in IT = No. of records in HD

Sparse Index

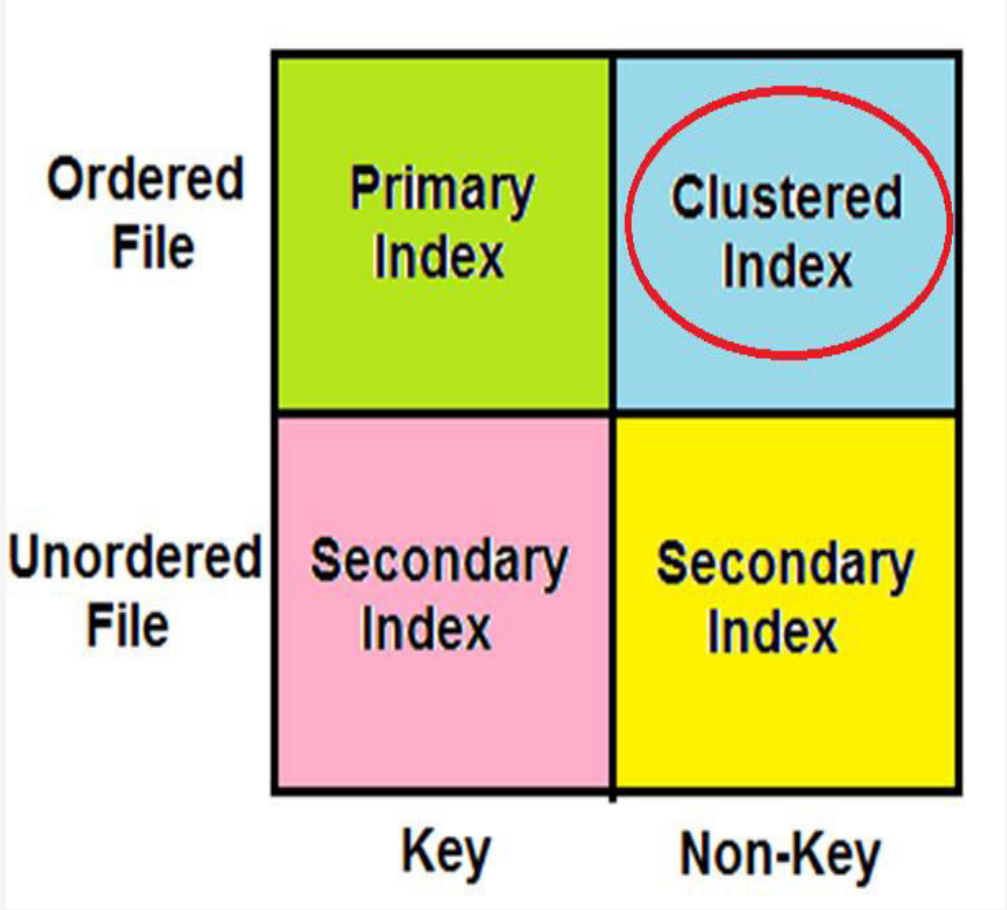


- Sparse index contains only the anchor records
- To locate a record, we find the index record with the largest search key value <= search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along with the pointers in the file (sequentially) until we find the desired record.

No. of records in IT = No. of blocks in HD

Time Complexity = $\log_2 N + 1$

Clustered Indexing



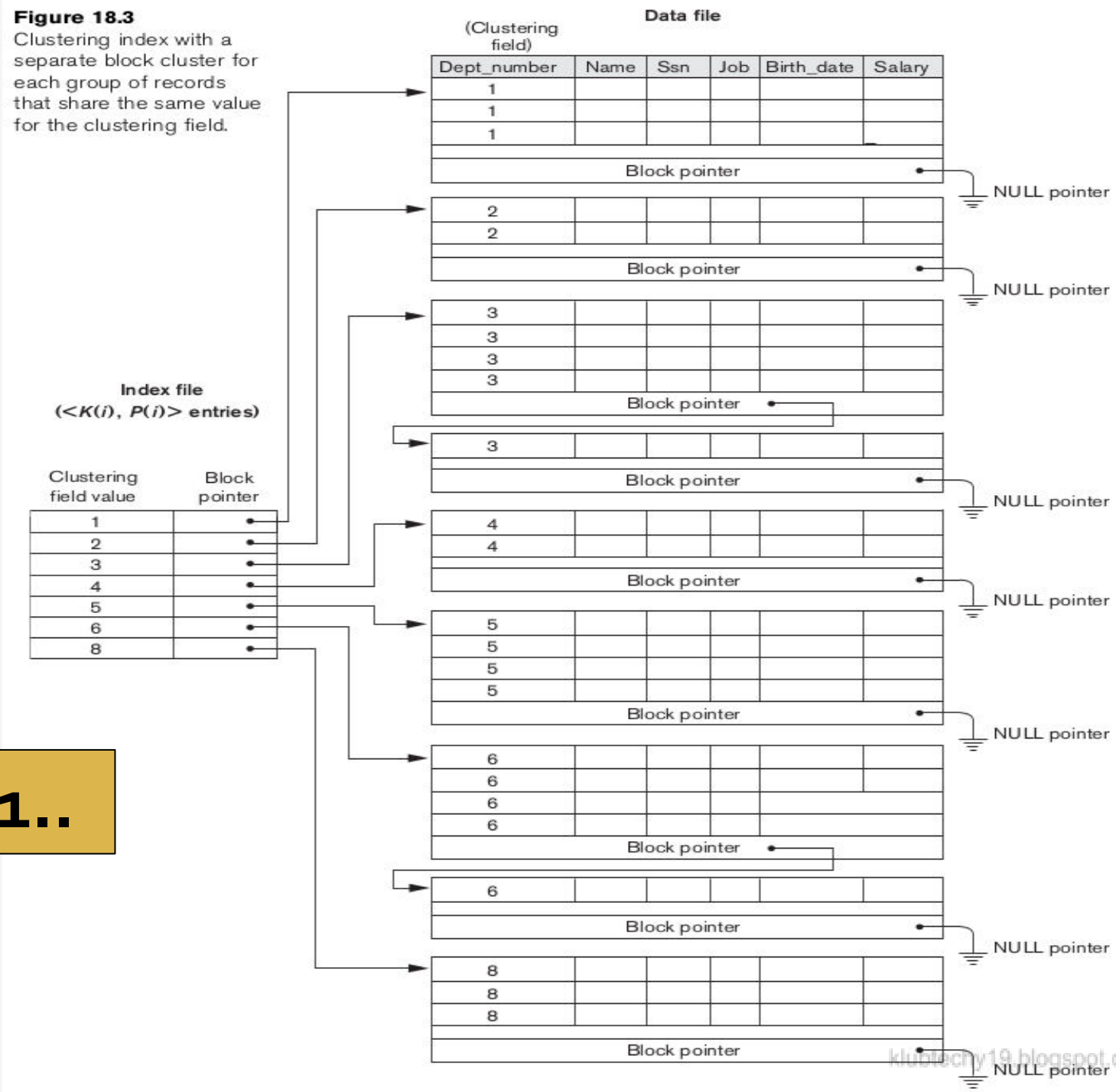
- Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.
- In some cases, the index is created on non-primary key columns which may not be unique for each record.
- In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as the clustering index.
- Basically, records with similar characteristics are grouped together and indexes are created for these groups.

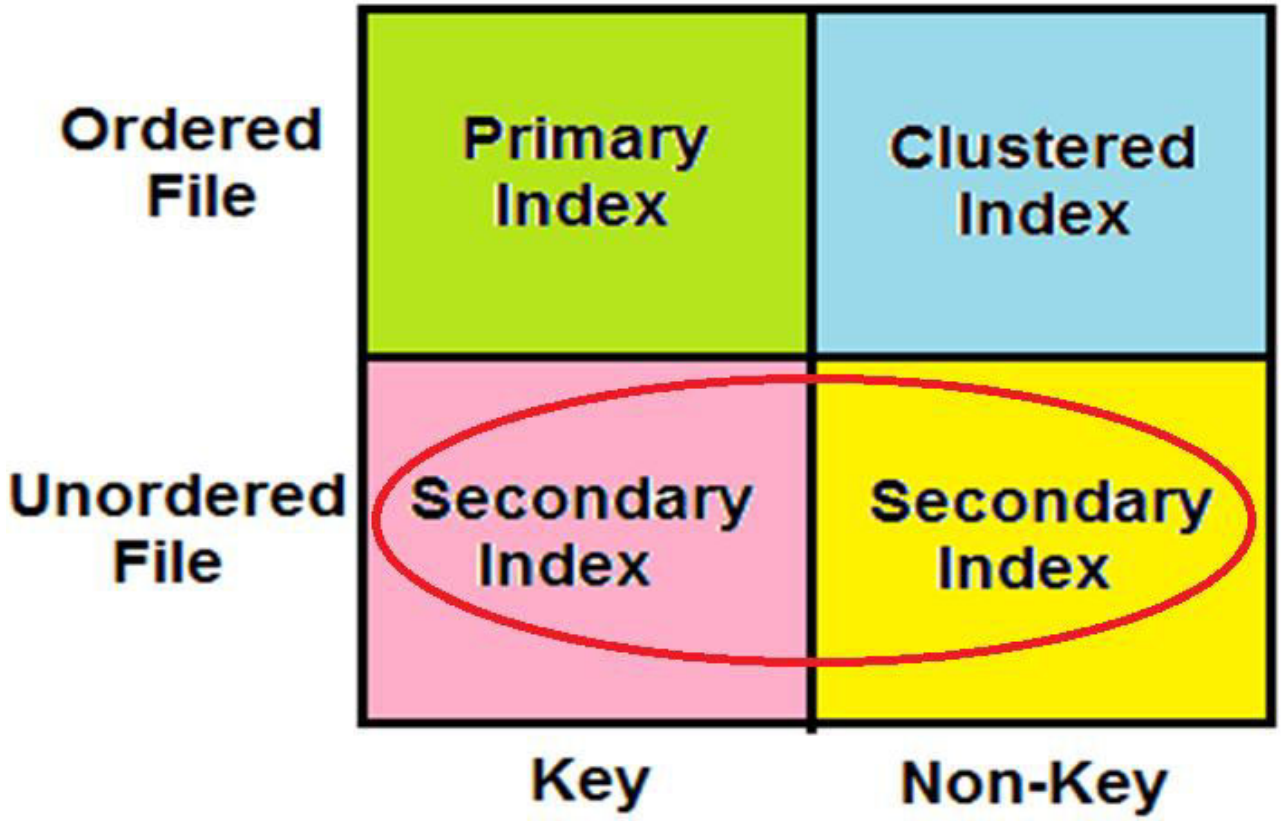
Contains block pointer which points to the next block data with the same clustering field value.

Searching criteria is little bit increased.

Uses Sparse index

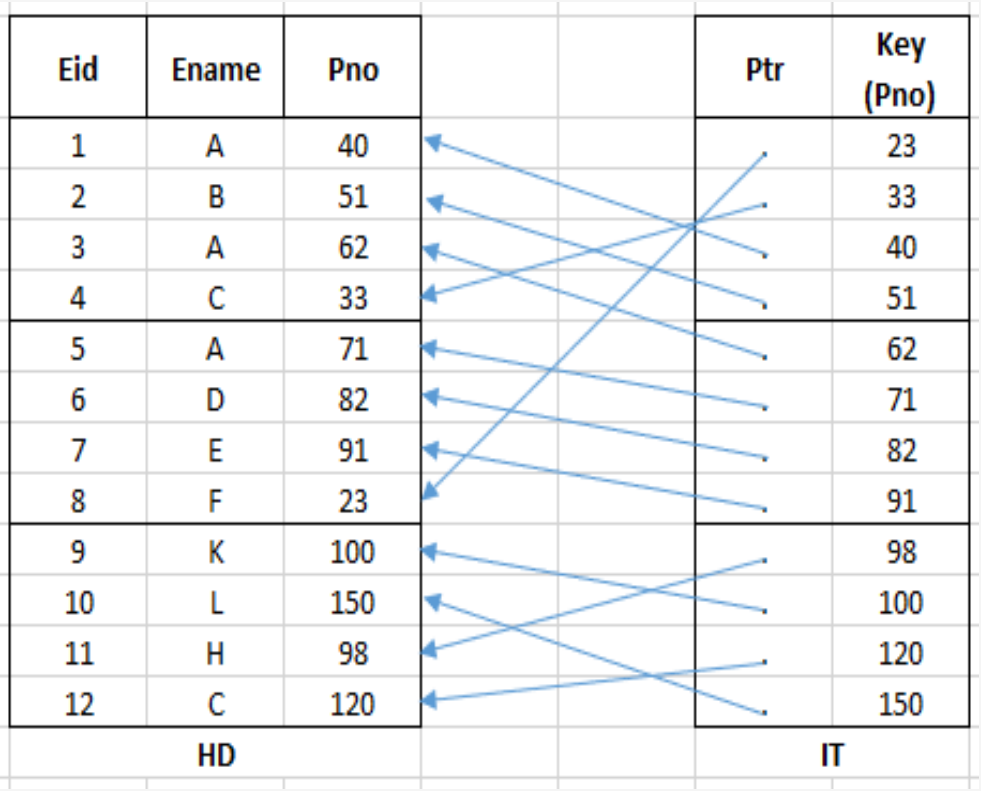
Time Complexity = $\log_2 N + 1 + 1$..





SECONDARY INDEXING

Unordered file with secondary key



Eid	Ename	Pno
1	A	40
2	B	51
3	A	62
4	C	33
5	A	71
6	D	82
7	E	91
8	F	23
9	K	100
10	L	150
11	H	98
12	C	120

HD

Ptr	Key (Pno)
	23
	33
	40
	51
	62
	71
	82
	91
	98
	100
	120
	150

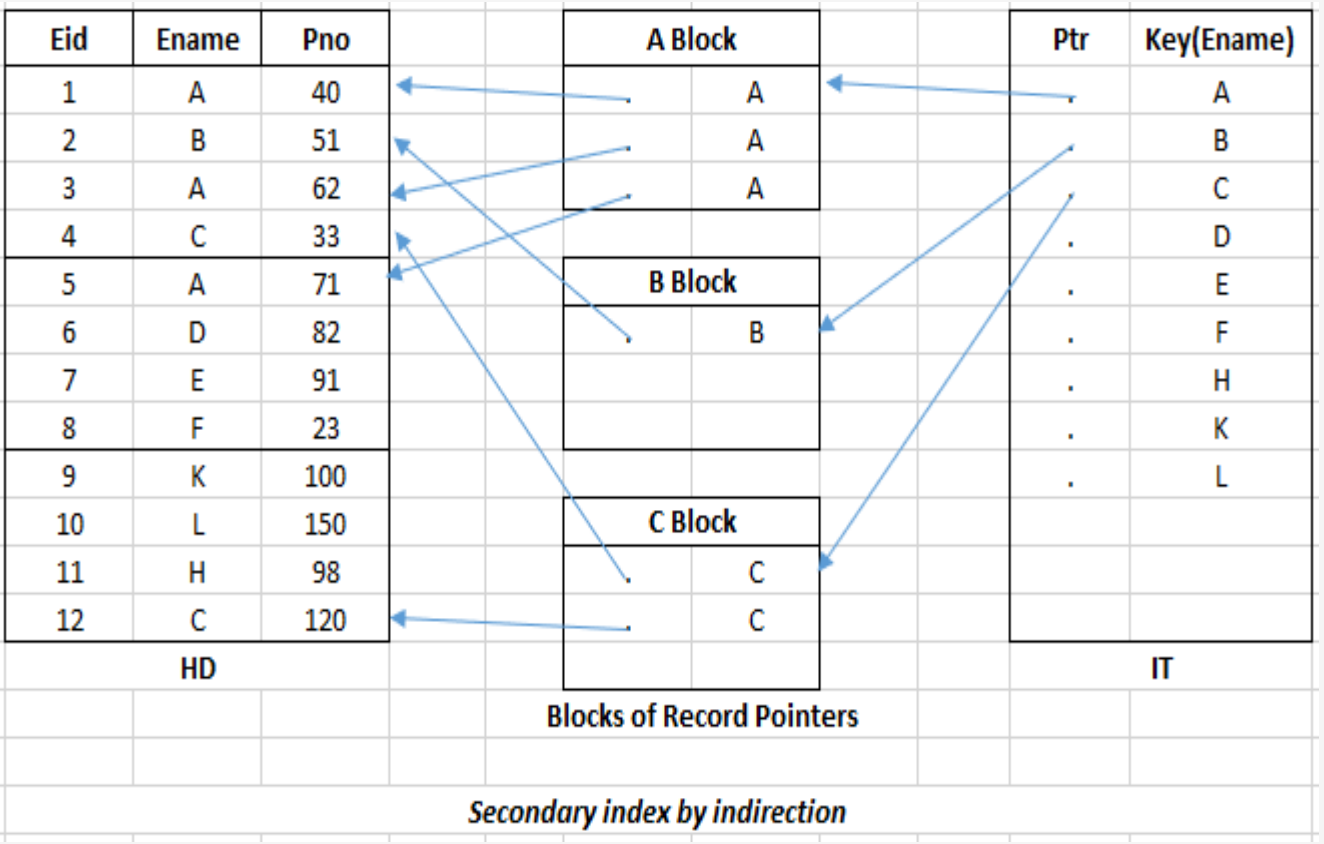
IT

Secondary Index Example

- File is ordered on Eid(Primary Key)
- Search to be done using Pno
- So, Index table will maintain Pno as a key and in ordered.

Time Complexity = $\log_2 N + 1$

Unordered file with Non-key



Secondary Index Example

- Search done by Ename(Non-key)
- Index file contains Ename as key and is ordered.
- Maintains intermediate index layer which contains block of record pointers.
- Pointer in IT points to a particular block and the record pointers in that block will point to the record in HD.

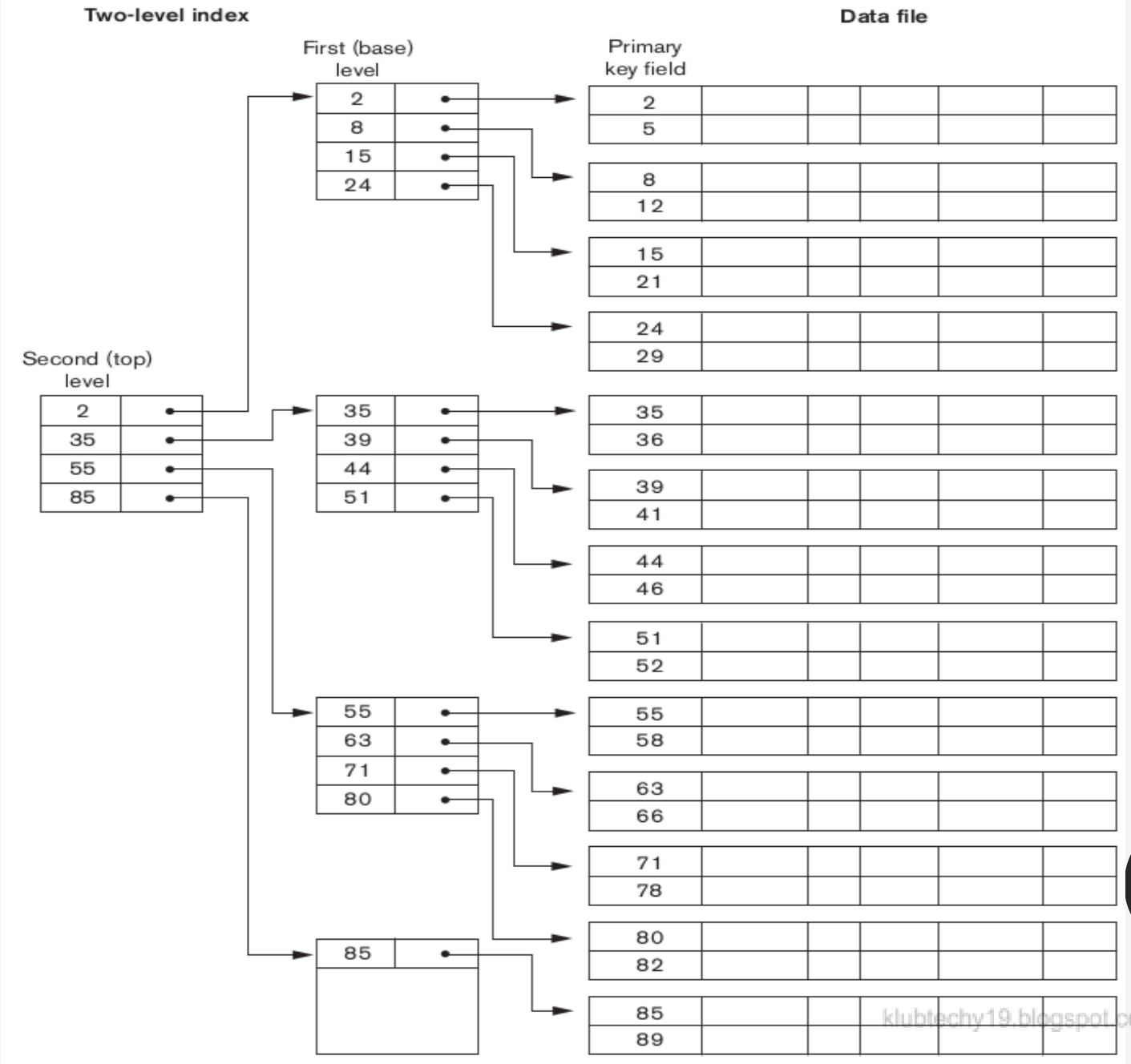
Time Complexity = $\log_2 N + 1 + 1$

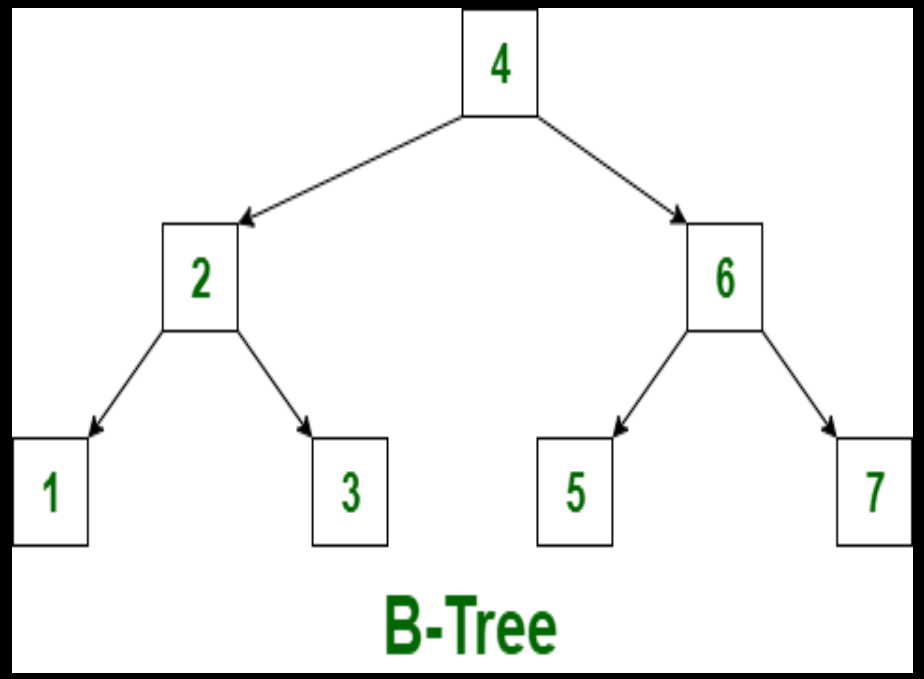
Multilevel Indexing

Creating an index on the index

Radhika Rani Chintala

Figure 18.6
A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.



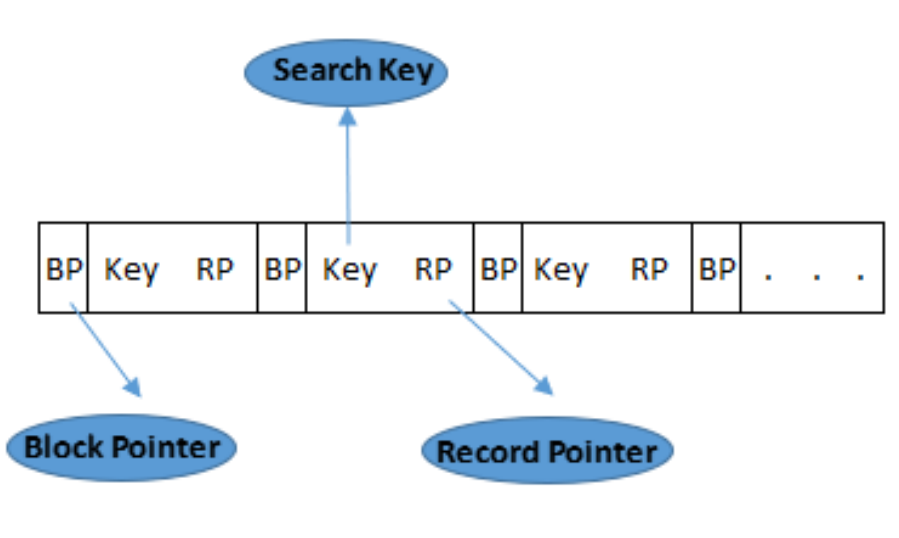


Dynamic Multilevel Indexing using B-Trees & B+ Trees

- Balanced Tree
- In multilevel indexing, inserting and deleting a record is difficult, as the corresponding entries in index tables also need to be changed.
- B-Trees makes these tasks simple.
- Elements are in sorted order

Radhika Rani Chintala

B-Tree



A B tree of order m contains the following properties:

- Every node in a B-Tree contains at most m-1 keys and m children.
- Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
- The root nodes must have at least 2 nodes.
- All leaf nodes must be at the same level.
- It is not necessary that, all the nodes contain the same number of children but, each node must have m/2 number of nodes.

Radhika Rani Chintala

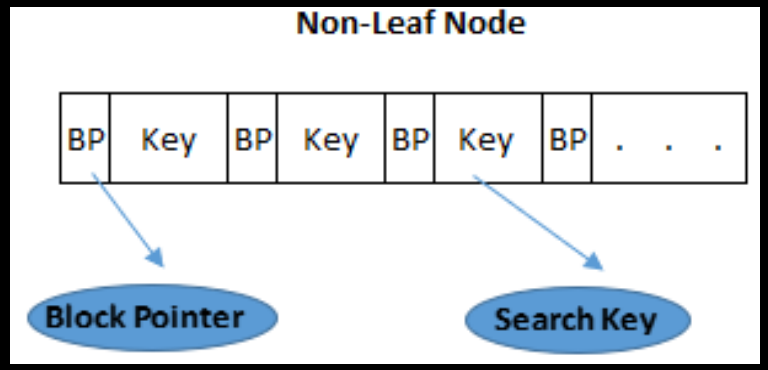
- B-Tree Example.docx

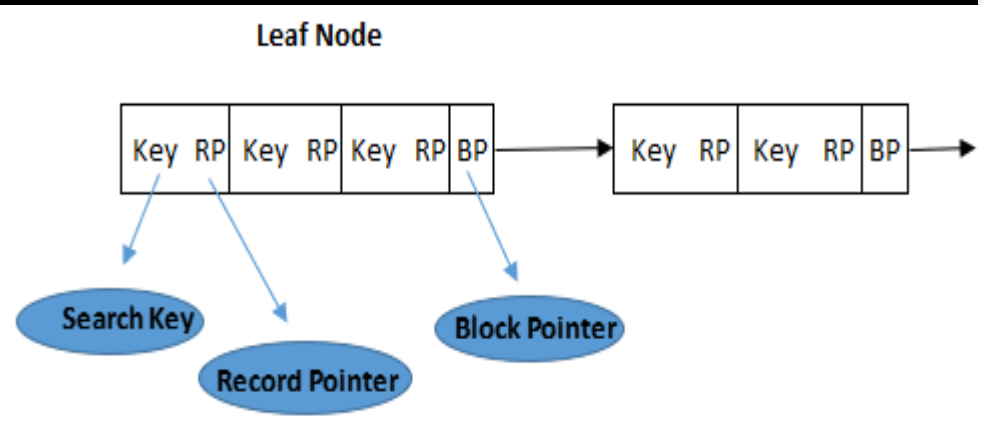
  - Insertion
  - Search
  - Deletion

B-Tree Example

Radhika Rani Chintala

B+ Tree





- B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.
- In B Tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.
- The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.

Radhika Rani Chintala

Difference between B Tree and B+ Tree

	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.
4	Deletion of internal nodes are so complicated and time consuming.	Deletion will never be a complexed process since element will always be deleted from the leaf nodes.
5	Leaf nodes can not be linked together.	Leaf nodes are linked together to make the search operations more efficient.

Radhika Rani Chintala

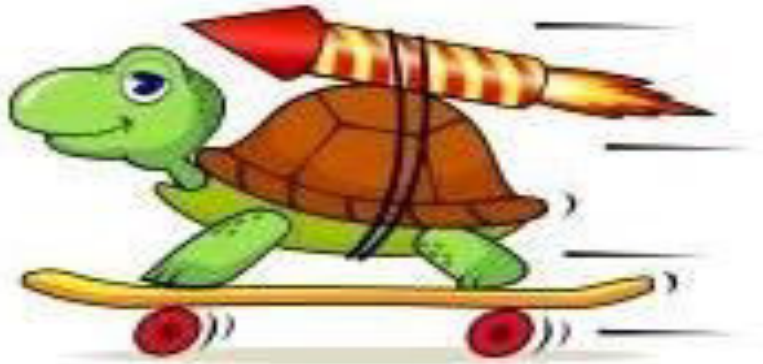
B+ Tree Example

Construct B+ tree of order 3 for the following elements.

5, 9, 1, 10, 12, 13, 2, 3, 4

Radhika Rani Chintala

Why Indexing is important?



THE END

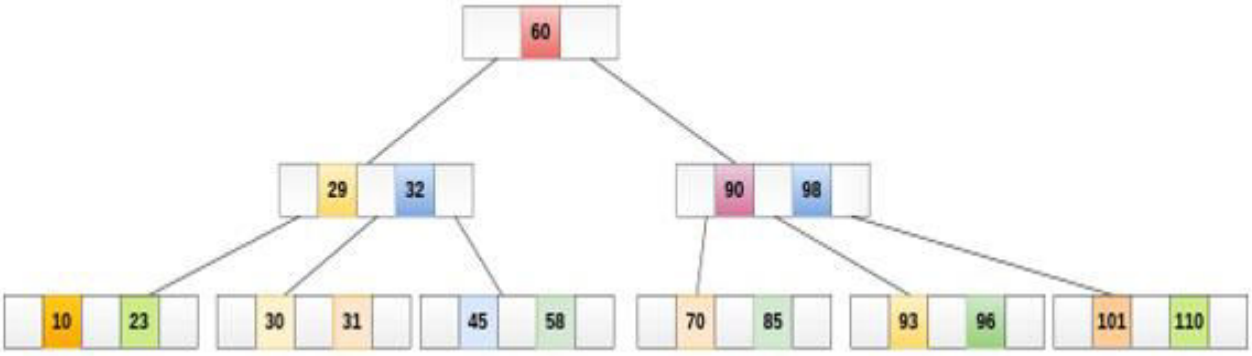
Radhika Rani Chintala

B- Tree Example

A B tree of order 4 is shown in the following image.

No. of Children = 4

No. of Keys = 4 – 1 = 3



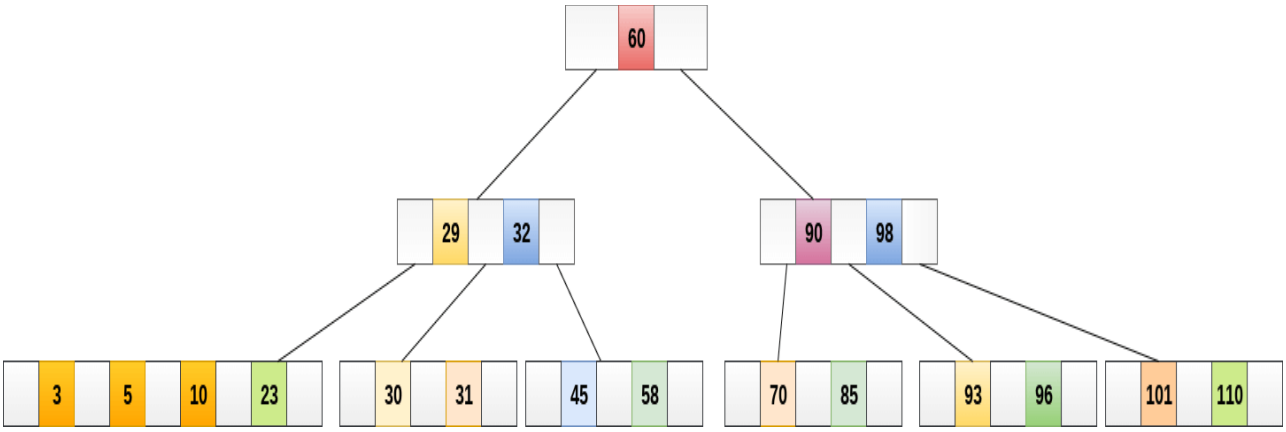
Searching :

For example, if we search for an item 31 in the following B Tree. The process will something like following :

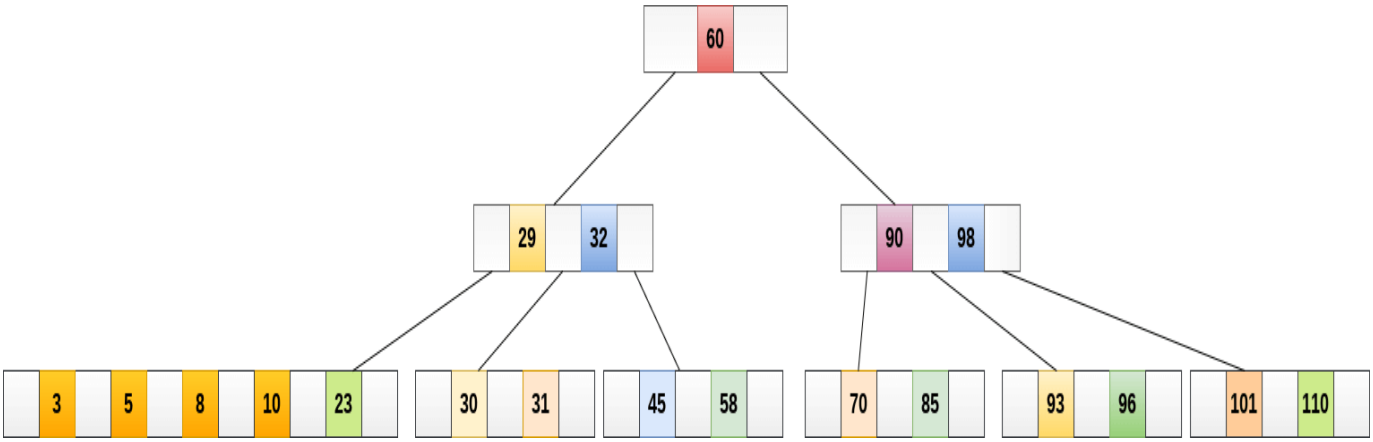
1. Compare item 31 with root node 60. since 31 < 60 hence, move to its left sub-tree.
2. Since, 29<31<32, traverse right sub-tree of 29.
3. 31>30, move to right. Compare 31.
4. match found, return.

Inserting :

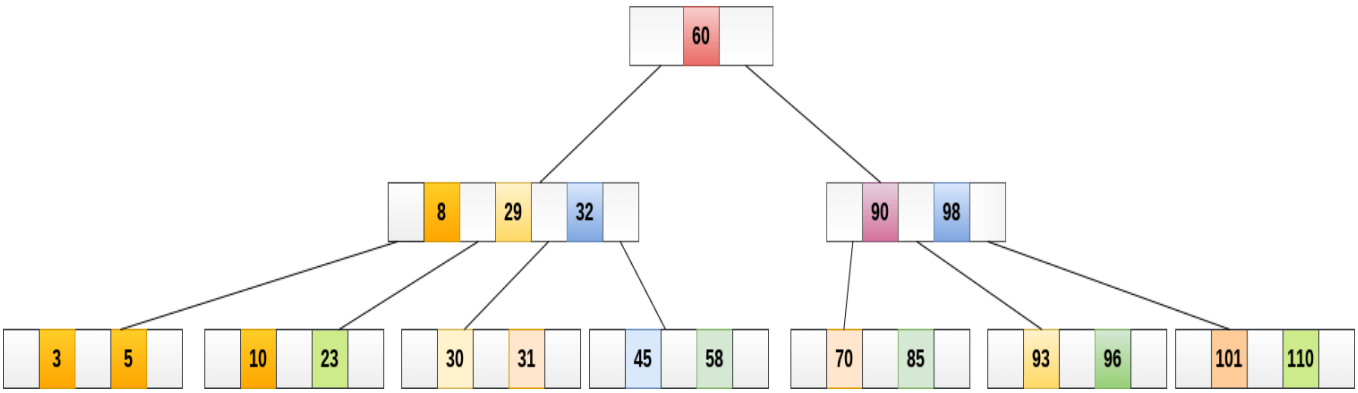
Insert the node 8 into the B Tree of order 5 shown in the following image.



8 will be inserted to the right of 5, therefore insert 8.

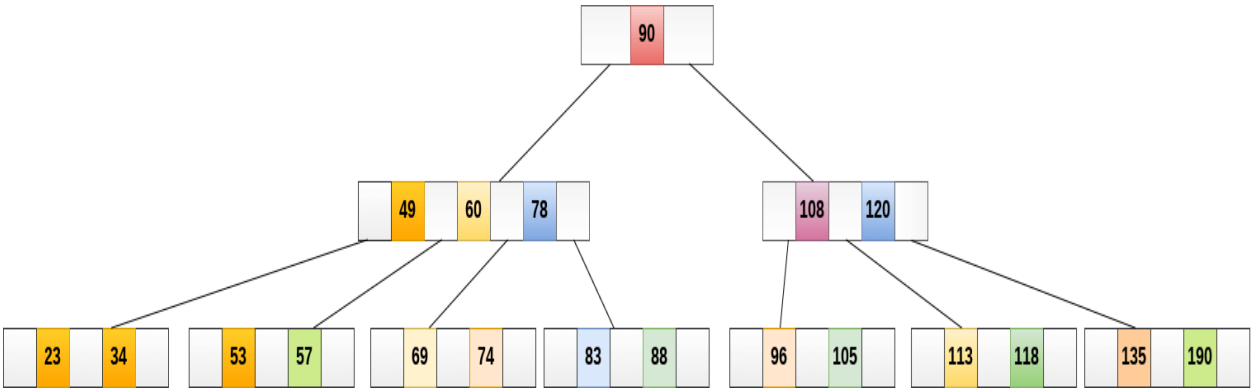


The node, now contain 5 keys which is greater than (5 -1 = 4) keys. Therefore split the node from the median i.e. 8 and push it up to its parent node shown as follows.

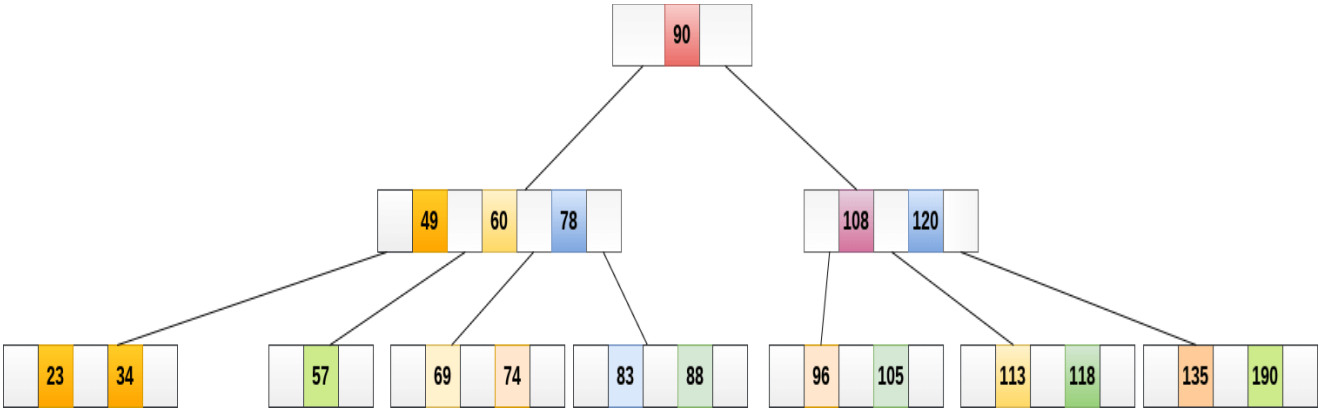


Deletion

Delete the node 53 from the B Tree of order 5 shown in the following figure.

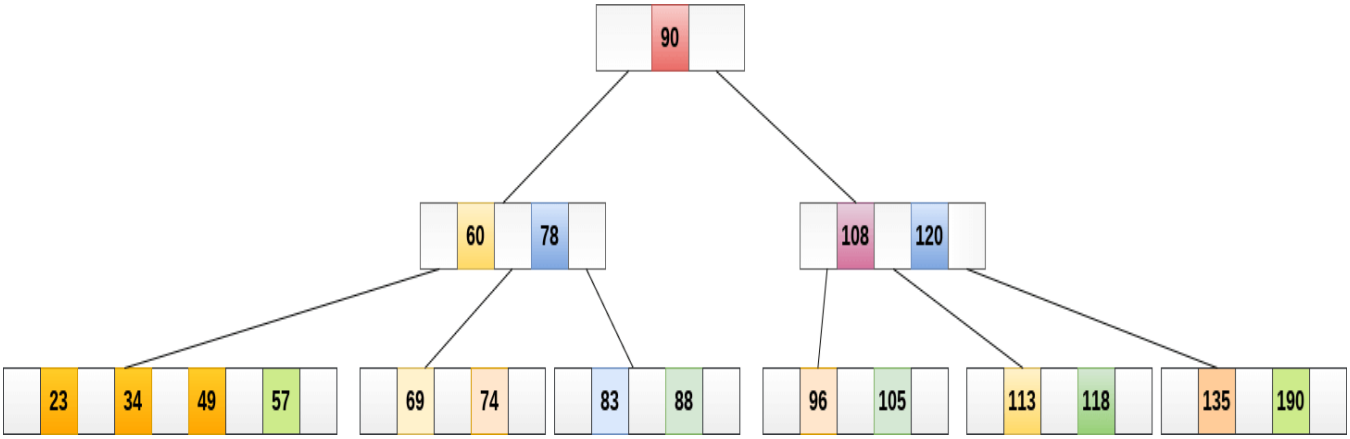


53 is present in the right child of element 49. Delete it.



Now, 57 is the only element which is left in the node, the minimum number of elements that must be present in a B tree of order 5, is 2. it is less than that, the elements in its left and right sub-tree are also not sufficient therefore, merge it with the left sibling and intervening element of parent i.e. 49.

The final B tree is shown as follows.



Hashing

Agenda

- Introduction to hashing
- Types of hashing
- Indexing Vs Hashing

Introduction to Hashing

- For a huge database structure, it's tough to search all the index values through all its level and then you need to reach the destination data block to get the desired data.

- Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

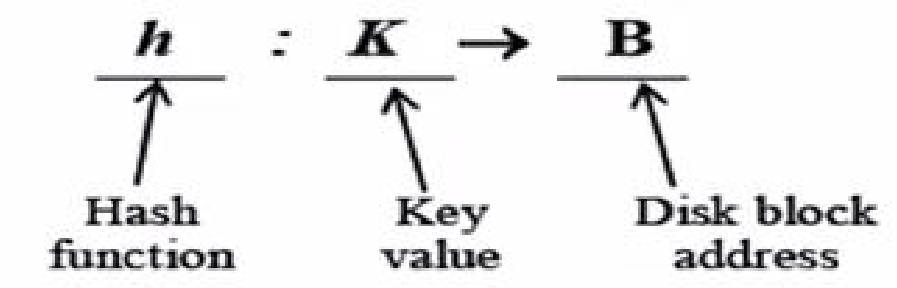
- Faster method to search data

Important Terminologies

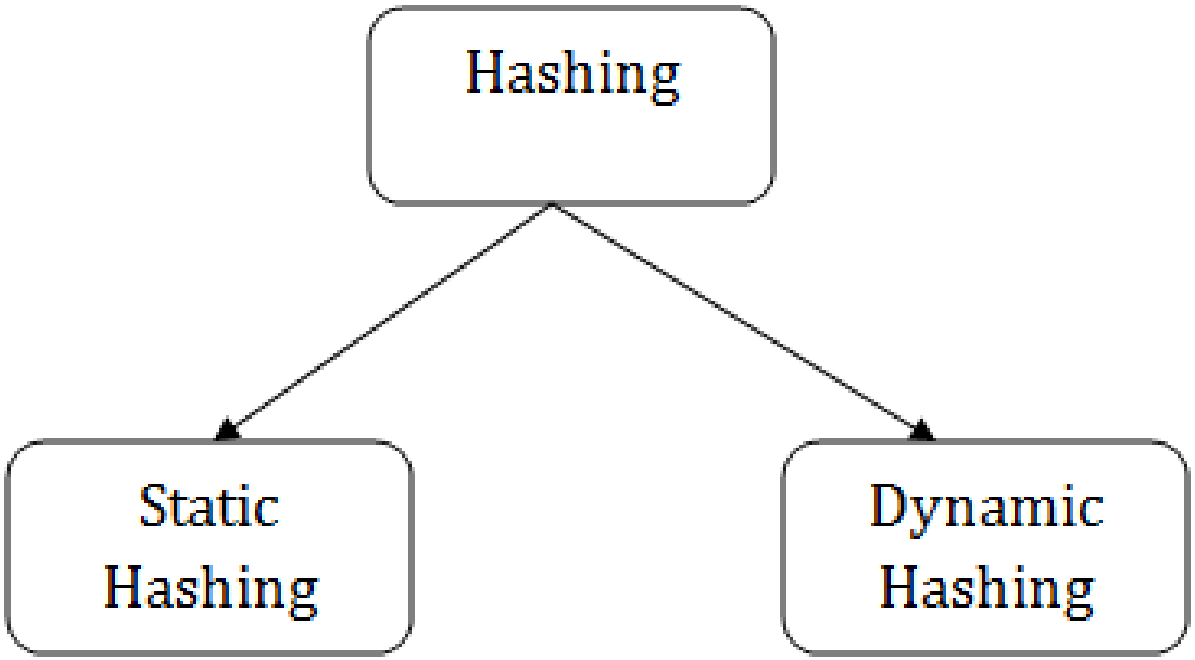
- **Data bucket** – Data buckets are memory locations where the records are stored.

- **Hash function**: A hash function, is a mapping function which maps all the set of search keys to the address where actual records are placed.
- This hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc.

- **Bucket Overflow**: The condition of bucket-overflow is called collision.



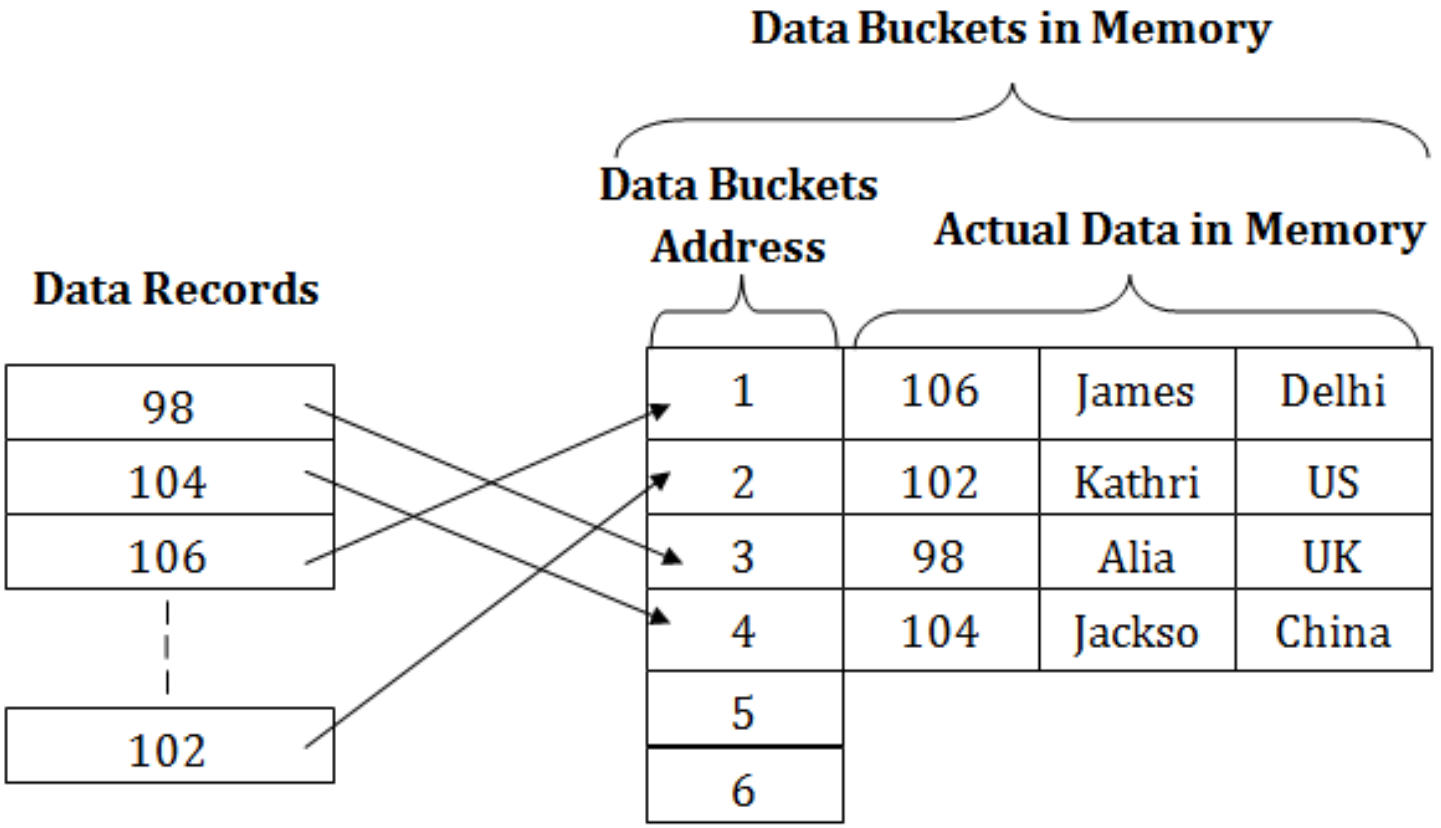
Types of Hashing



Static Hashing

- In static hashing, when a search-key value is provided, the hash function always computes the same address.
- For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function.
- The number of buckets provided remains unchanged at all times.

Hash function (h) = k mod 5



Operations

- **Insertion** − When a record is required to be entered using static hash, the hash function **h** computes the bucket address for search key **K**, where the record will be stored.

- Bucket address = h(K)

- **Search** − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.

- **Delete** − This is simply a search followed by a deletion operation.

Collision

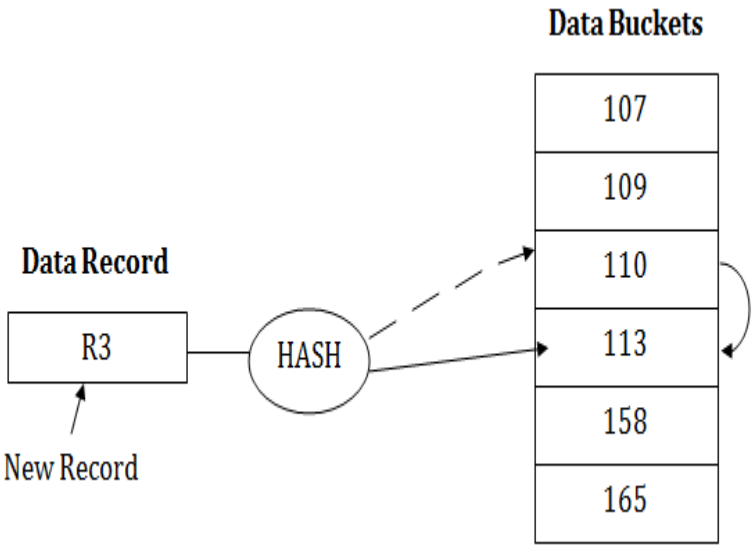
- The condition of bucket-overflow is known as collision.
- If we want to insert some new record into the file but the address of a data bucket generated by the hash function is not empty, or data already exists in that address. This situation in the static hashing is known as **bucket overflow**.
- This is a critical situation in this method.
- To overcome this situation, there are two methods namely:
  - **Open Hashing or Linear Probing**
  - **Closed Hashing or Overflow Chaining**

Collision Resolution

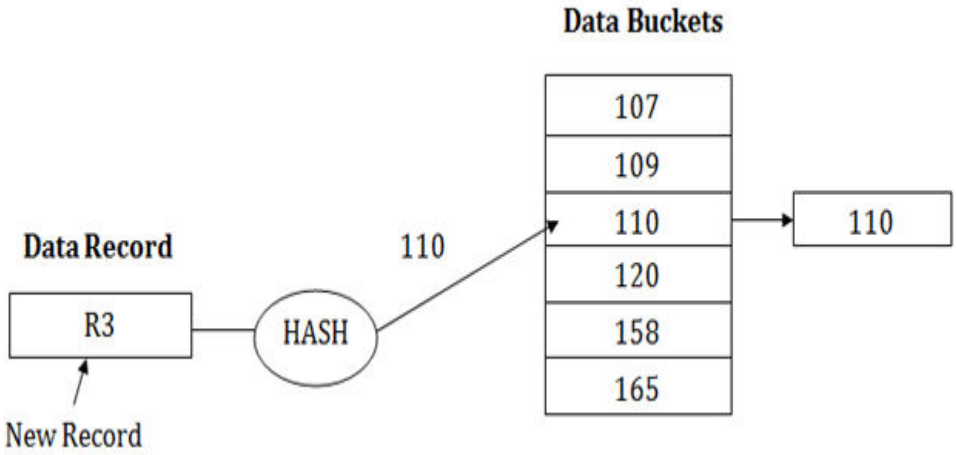
- **Linear Probing** − When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called **Open Hashing**.

- **Overflow Chaining** − When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called **Closed Hashing**.

Linear Probing



Overflow Chaining



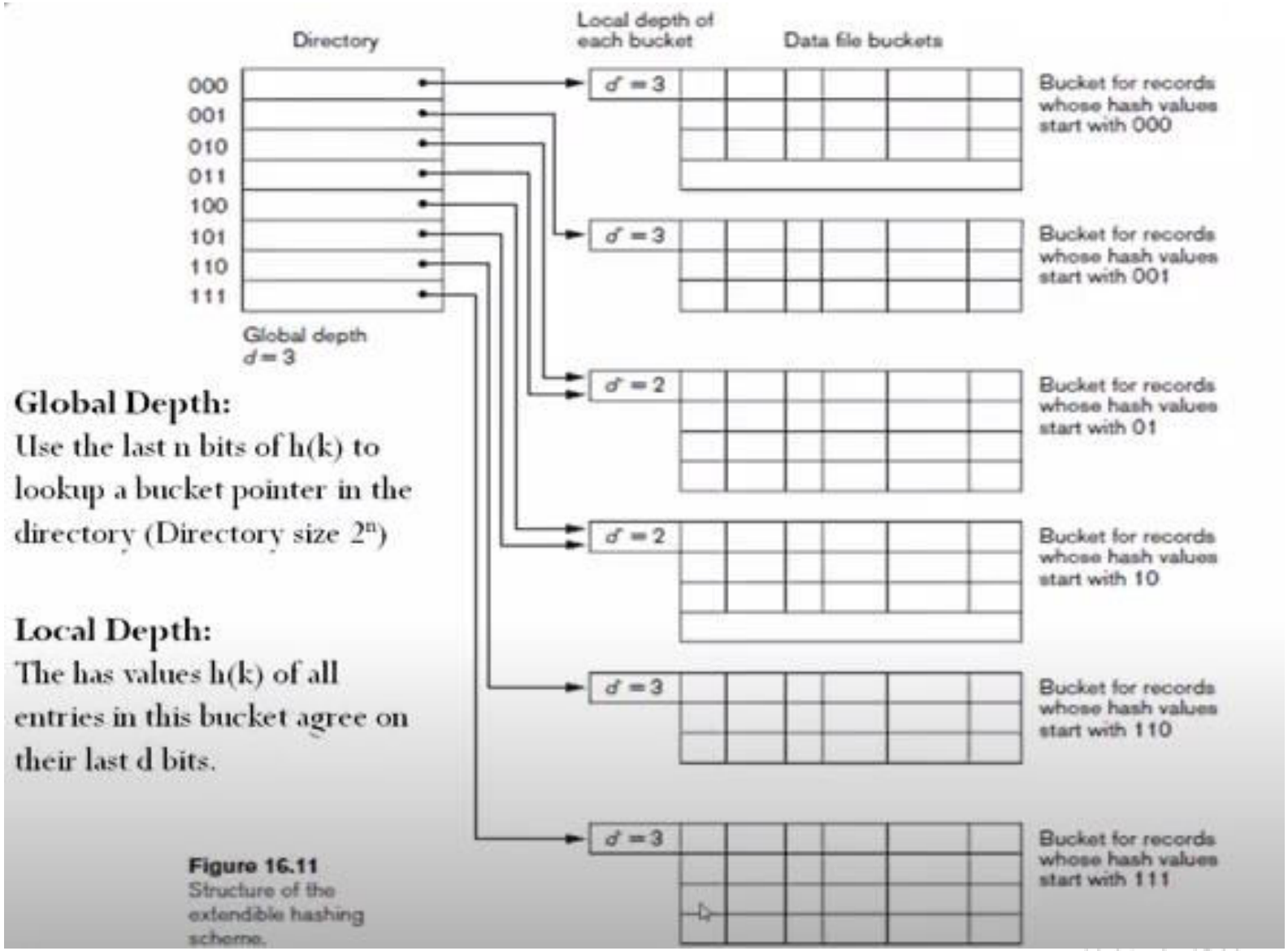
Dynamic Hashing

- The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks.
- Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand.
- Each result of hash function is represented in binary, called Hash value.
- Records are distributed among the buckets based on the values of the leading bits in their hash value.
- Types
  - **Extendible Hashing**
  - **Linear Hashing**

Extendible Hashing

- This scheme stores a directory structure in addition to the file.
- This access structure is based on the result of the hash function to the search key.
- Advantage: No additional space is wasted towards the allocations of new records.
- Overhead: Directory structure need to be searched before the buckets are accessed.

Extendible Hashing..



Global Depth:
Use the last n bits of h(k) to lookup a bucket pointer in the directory (Directory size 2^n)

Local Depth:
The has values h(k) of all entries in this bucket agree on their last d bits.

Figure 16.11
Structure of the extendible hashing scheme.

Example

- Hashing Example.docx

Linear Hashing

- No directory structure is used.
- Instead of single hash function, multiple hash functions are used.
- When collision occurs with one hash function, the bucket that overflows is split into two and the records in the original bucket are distributed among two buckets using the next hash function **$h_{i+1}(k)$.**

Indexing Vs Hashing

INDEXING	HASHING
A data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done	An effective technique to calculate the direct location of a data record on the disk without using index structure
Uses data reference that holds the address of the disk block with the value corresponding to the key	Uses mathematical functions called hash functions to calculate direct locations of data records on the disk
Does not work well for large databases	Works well for large databases

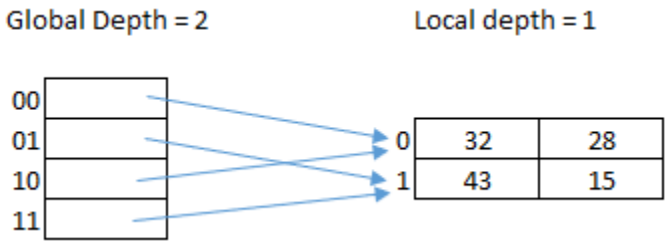
Extendible Hashing Example

Key values: 32, 28, 43, 15, 66, 27

Hash Function h(k) = k mod 10

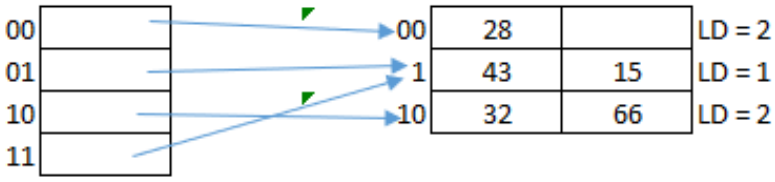
Bucket size: 2

Key	32	28	43	15	66	27
Hash value	2	8	3	5	6	7
In Binary	0010	1000	0011	0101	0110	0111



Insert 66

No space in bucket. So expand the by allocating new bucket. Local depth of that bucket becomes 2.



Insert 27

No space in bucket. So expand the by allocating new bucket. Local depth of that bucket becomes 2.