Database Management System
Module 3

Lect 1 : SQL- Data Manipulation Language contn…

MODULE 3: **SQL DML (Data Manipulation Language), Physical Data Organization**

SYLLABUS

- SQL DML (Data Manipulation Language)
  - SQL queries on single and multiple tables, Nested queries (correlated and non-correlated), Aggregation and grouping, Views, assertions, Triggers, SQL data types.
- Physical Data Organization
  - Review of terms: physical and logical records, blocking factor, pinned and unpinned organization. Heap files, Indexing, Singe level indices, numerical examples, Multi-level-indices, numerical examples, B-Trees & B+-Trees (structure only, algorithms not required), Extendible Hashing, Indexing on multiple keys – grid files

Data Manipulation Language

- DML stands for Data Manipulation Language.
- It is a language used for selecting, inserting, deleting and updating data in a database.
- It is used to retrieve and manipulate data in a relational database.
- There are three basic constructs which allow database program and user to enter data and information are:
  - INSERT
  - UPDATE
  - DELETE

Basic Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the SELECT statement

SELECT <attribute list>
FROM <table list>
WHERE <condition>

- <attribute list>
  - is a list of attribute names whose values are to be retrieved by the query
- <table list>
  - is a list of the relation names required to process the query
- <condition>
  - is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

The select Clause

- The select clause lists the attributes desired in the result of a query

- Example: find the names of all instructors:

```
select name
from instructor
```

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., Name = NAME = name

The select Clause

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

```
SELECT name from `instructor`
```

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

The select Clause

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword distinct after select.
- Find the department names of all instructors, and remove duplicates

```
select distinct dept_name
from instructor
```

- The keyword all specifies that duplicates should not be removed.

```
select all dept_name
from instructor
```

The select Clause

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

```
SELECT DISTINCT `dept_name` FROM `instructor`
```

dept_name
Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics

The select Clause

- An asterisk in the select clause denotes "all attributes"

```
select *
from instructor
```

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

The select Clause

- The select clause can contain arithmetic expressions involving the operation, +, –, □, and /, and operating on constants or attributes of tuples.
  - The query:

```
select ID, name, salary/12
from instructor
```

would return a relation that is the same as the instructor relation, except that the value of the attribute salary is divided by 12.

  - Can rename "salary/12" using the as clause:

```
select ID, name, salary/12  as monthly_salary
```

The select Clause

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

```
SELECT `ID`,`name`,`salary`/12 FROM `instructor`
WHERE 1
```

ID	name	`salary`/12
10101	Srinivasan	5416.666667
12121	Wu	7500.000000
15151	Mozart	3333.333333
22222	Einstein	7916.666667
32343	El Said	5000.000000
33456	Gold	7250.000000
45565	Katz	6250.000000
58583	Califieri	5166.666667
76543	Singh	6666.666667
76766	Crick	6000.000000
83821	Brandt	7666.666667
98345	Kim	6666.666667

The where Clause

- The where clause specifies conditions that the result must satisfy

- To find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

  - To find all instructors in Comp. Sci. dept with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.'  and salary > 80000
```

The where Clause

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

```
SELECT `name` FROM `instructor` WHERE `dept_name`="Comp. Sci."
```

name
Srinivasan
Katz
Brandt

SQL Set Operation

Types of Set Operation

- Union
- UnionAll
- Intersect
- Minus

1. Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

SELECT * FROM First UNION SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

2. Union All

- Union All operation is equal to the Union operation.
- It returns the set without removing duplication and sorting the data.

Syntax

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

- **Example:** Using the above First and Second table.

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.

- **Syntax**

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example:

- Using the First and Second table.

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

ID	NAME
3	Jackson

Minus

- It combines the result of two SELECT statements.
- Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry

Database Management System
Module 3

Lect 2 :
-> SQL- Aggregate Function,

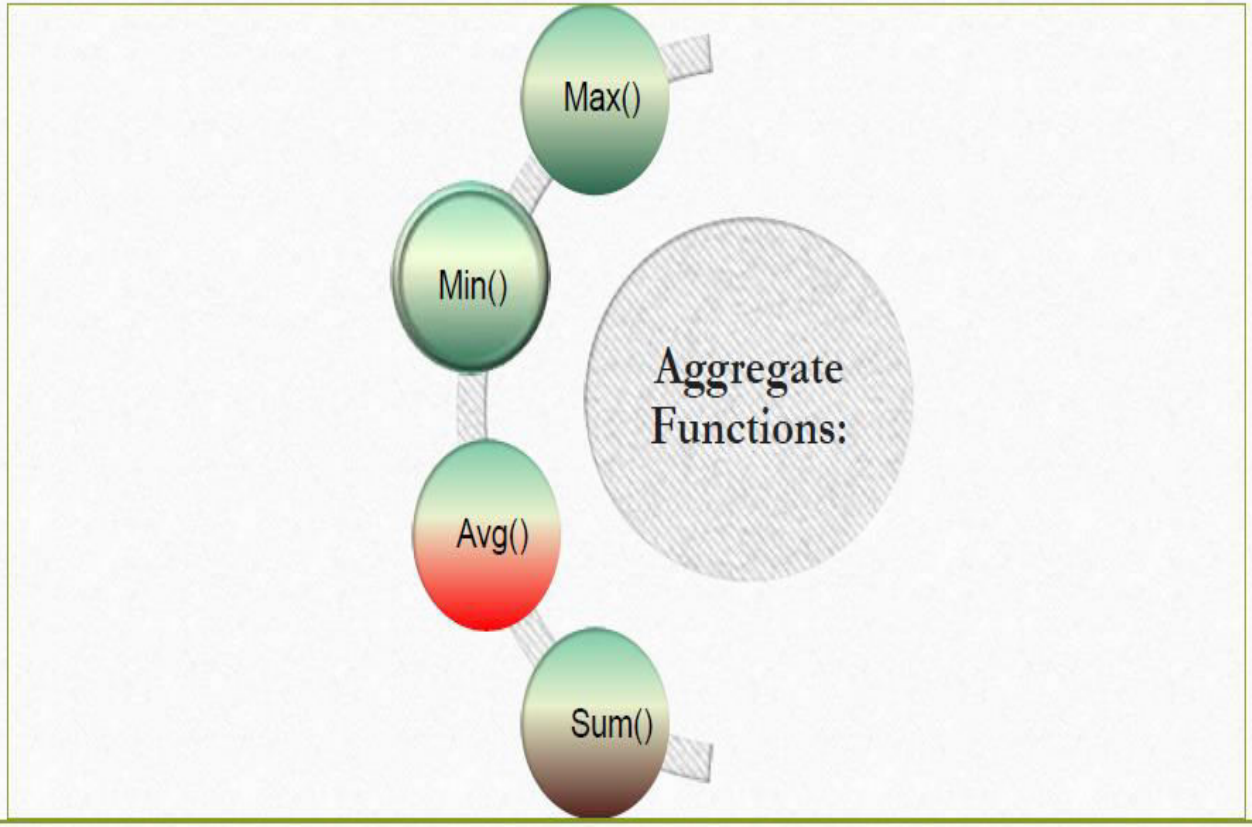
SQL AGGREGATE FUNCTION

Aggregate Function:

Aggregate functions are functions that take a collection of values as input and return a single value.

- Behavior of Aggregate Functions:
  - Operates - on a single column
  - Return - a single value.

Types of SQL Aggregate Functions:



Input to Aggregate Function

- SUM and AVG :
  - ➔ Operates only on collections of numbers .
- MIN , MAX and COUNT
  - ➔ Operates on collection of numeric and non-numeric data types.

❑ Each function eliminates NULL values and operates on Non- null values.

Staff

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	24000.00	Manager
SL102	David	Ford	12000.00	Project Manager
SL103	Ann	Beech	12000.00	Project Manager
SL104	Mary	Howe	9000.00	Project Manager

USE OF SUM()

Returns: The sum of the values in a specified column.

Example: Find the total/sum of the Managers salary

Query: select sum(salary) from staff where position = manager

Result:

sum_salary
54000.00

USE OF AVG()

Returns: The average of the values in a specified column.

Example: Find the average of the Project Managers salary .

Query:

```
Select avg(salary) from staff where
  position='project manager';
```

Result :

avg_salary
11000.00

MIN() and MAX()

Returns: MIN() returns the smallest value of a column.
MAX() returns the largest value of a column.

Example: Find the minimum and maximum staff salary.

```
select min(salary),max(salary) from staff;

MIN(SALARY) MAX(SALARY)
----------- -----------
      30000       42000
```

- It is possible to rename the attribute name using AS cluase

```
select min(salary) AS salarymin ,max(salary) AS salarymax from staff;

 SALARYMIN  SALARYMAX
---------- ----------
     30000      42000
```

USE OF COUNT()

Returns: The number of values in the specified column.

Example: Count number of staffs who are Manager.

```
Query: SELECT  COUNT(sno) AS sno_count
                FROM Staff
       WHERE   position = 'Manager';
```

Result:

sno_count
2

Use of COUNT() and SUM()

Example: Find the total number of Managers and the sum of there salary.

```
select count(sno), sum(salary) from staff
  where position='manager';
```

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	24000.00	Manager

- Result:

sno_count	sum_salary
2	54000.00

Database Management System
Module 3

Lect 3 :
-> SQL- ORDER BY GROUP BY & HAVING CLAUSE,

ORDER BY, GROUP BY & HAVING CLAUSE

The SQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the columns in ascending or descending order.
- It sorts the records in ascending order by default.
- To sort the records in descending order, use the DESC keyword

Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

NAME	CITY	COUNTRY
Alfred	berlin	germany
Anna	Mcity	mexico
Antonio	mcity	mexico
james	london	uk
christina	beiging	china

Eg selects all customers from the "Customers" table, sorted by the "Country" column

Query : select * from customer order by country;

NAME	CITY	COUNTRY
christina	beiging	china
Alfred	berlin	germany
Anna	Mcity	mexico
Antonio	mcity	mexico
james	london	uk

selects all customers from the "Customers" table, sorted by the "Country" column in descending

select * from customer order by country desc;

NAME	CITY	COUNTRY
james	london	uk
Anna	Mcity	mexico
Antonio	mcity	mexico
Alfred	berlin	germany
christina	beiging	china

The SQL GROUP BY Statement

- The GROUP BY statement is used to to arrange identical data into groups
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

NAME	CITY	COUNTRY
Alfred	berlin	germany
Anna	Mcity	mexico
Antonio	mcity	mexico
james	london	uk
christina	beiging	china

Eg :lists the number of customers in each country

QUERY

```
select count(name), country from customer group
   by (country);
```

COUNT(NAME)	COUNTRY
1	uk
1	germany
2	mexico
1	china

Staff

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	24000.00	Manager
SL102	David	Ford	12000.00	Project Manager
SL103	Ann	Beech	12000.00	Project Manager
SL104	Mary	Howe	9000.00	Project Manager

Eg2 : List the minimum salary held by each positions

Query : select position, min(salary) from staff group by(position);

Result:

Position	min(salary)
Manager	24000
Project Manager	9000

Staff

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	24000.00	Manager
SL102	David	Ford	12000.00	Project Manager
SL103	Ann	Beech	12000.00	Project Manager
SL104	Mary	Howe	9000.00	Project Manager

Eg :2 select the number of employee working in various positions
Query : select position,count(fname) from staff GROUPBY(position);
Result:

Position	count(fname)
Manager	2
Project manager	3

HAVING Clause

- It is used to state a condition that applies to groups
- It can be used in conjunction with GROUP BY clause

Staff

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	4000.00	Manager
SL102	David	Ford	12000.00	Project Manager
SL103	Ann	Beech	9000.00	Project Manager
SL104	Mary	Howe	9000.00	Project Manager

Eg :

select the name of the employee held in various positions whose salary is above 10,000

Query :

SELECT position,fname from staff GROUPBY(position) HAVING salary >10000;

Result:

Position fname

Manager John

Project manager David

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Ramesh	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	kaushik	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Q1) List the total amount of salary on each customer

Query : select name,sum(salary) from employee GROUP BY name;

Result

Name	sum(salary)
Ramesh	3500
Kaushik	8500
Hardrik	8500
Komal	4500
Muffy	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Q2) Display the name & age of employee according to alphabetic order of names

Query : select name, age from employee ORDER BY name;

Result

Name	age
Chaitali	25
Hardrik	27
Kaushik	23
Khilan	25
Komal	22
Muffy	24
Ramesh	32

Cust_id	Name	Country
1	John	Mexico
2	Stalin	France
3	Antonio	Mexico
4	Tom	Swedan
5	Alex	France
6	Aleena	France

Q3) List the number of customers in each country, Only include countries with more than 1 employees

Query : select count(cust_id), country from employee GROUP BY country HAVING count(cust_id)>1;

Result

Count(cust_id)	Country
2	Mexico
3	France

Cust_id	Name	Country
1	John	Mexico
2	Stalin	France
3	Antonio	Mexico
4	Tom	Swedan
5	Alex	France
6	Aleena	France

Q3) List the number of customers in each country,

Query : select count(cust_id), country from employee GROUP BY country

Result

Count(cust_id)	Country
2	Mexico
3	France
1	Swedan

Database Management System
Module 3

Lect 5 :
-> SQL- JOIN

Why SQL Join?

Eg : Get the name of the staff working in HR dept?

Soln : Combine/ Join table using attribute which are in common to both table

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

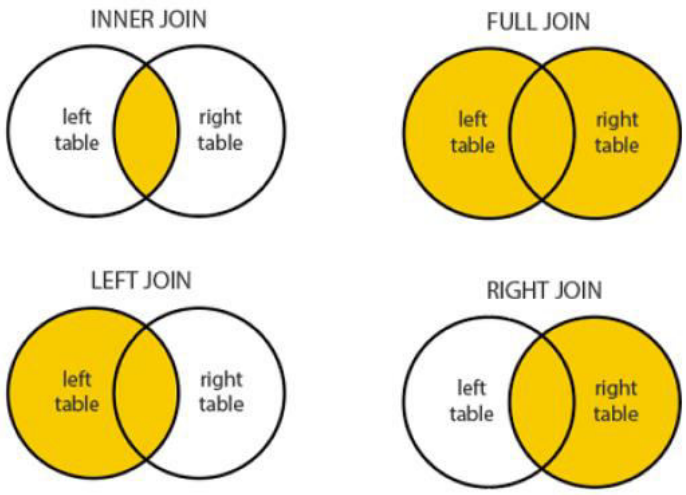
DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

SQL Join

- A SQL JOIN combines records from two tables.
- A JOIN locates related column values in the two tables.
- A query can contain zero, one, or multiple JOIN operations.

Four different types of JOINs

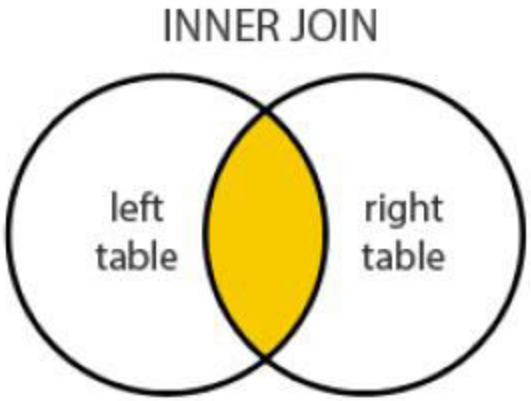
- INNER JOIN: Select records that have matching values in both tables.
- FULL (OUTER) JOIN: Selects all records that match either left or right table records.
- LEFT (OUTER) JOIN: Select records from the first (left-most) table with matching right table records.
- RIGHT (OUTER) JOIN: Select records from the second (right-most) table with matching left table record



1. INNER JOIN

- Select records that have matching values in both tables.
- The general syntax is

SELECT column-names FROM table-name1 INNER JOIN table name2 ON column-name1 = column-name2 WHERE condition



Eg : Get Name,Position, dno and dep name of all staffs

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

```
SELECT staff.name, staff.position, dep.dno,
  dep.dname FROM staff INNER JOIN dep ON
  staff.dno=dep.dno
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

Result:

Number of Records: 4

NAME	POSITION	DNO	DNAME
JOHN	MANAGER	D100	HR
SUSAN	MANAGER	D101	FINANCE
ANN	PROJECT MANAGER	D103	LOGISTICS
MARY	PROJECT MANAGER	D101	FINANCE

Alternately

```
SELECT staff.name, staff.position, dep.dno,
  dep.dname  FROM staff ,dep WHERE
  staff.dno=dep.dno;
```

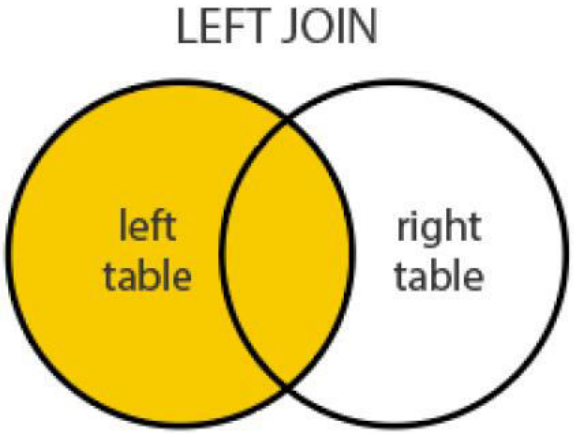
Result:

Number of Records: 4

NAME	POSITION	DNO	DNAME
JOHN	MANAGER	D100	HR
SUSAN	MANAGER	D101	FINANCE
ANN	PROJECT MANAGER	D103	LOGISTICS
MARY	PROJECT MANAGER	D101	FINANCE

2. LEFT OUTER JOIN/LEFT JOIN

- A LEFT JOIN performs a join starting with the first (left-most) table.
- Then, any matched records from the second table (right-most) will be included.
- LEFT JOIN and LEFT OUTER JOIN are the same.



- The general LEFT OUTER JOIN syntax is

SELECT column-names FROM table-name1 LEFT OUTER JOIN table-name2 ON column-name1 = column-name2 WHERE condition

Eg : Get Name,Position, dno and dep name of all staffs

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

```
SELECT staff.name,staff.position,dep.dno,dep.dname
  FROM staff LEFT JOIN dep ON
  staff.dno=dep.dno
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

Number of Records: 5

NAME	POSITION	DNO	DNAME
JOHN	MANAGER	D100	HR
SUSAN	MANAGER	D101	FINANCE
DAVID	PROJECT MANAGER	*null*	*null*
ANN	PROJECT MANAGER	D103	LOGISTICS
MARY	PROJECT MANAGER	D101	FINANCE

3. RIGHT OUTER JOIN/RIGHT JOIN

- A RIGHT JOIN performs a join starting with the second (right-most) table and then any matching first (left-most) table records.
- RIGHT JOIN and RIGHT OUTER JOIN are the same.

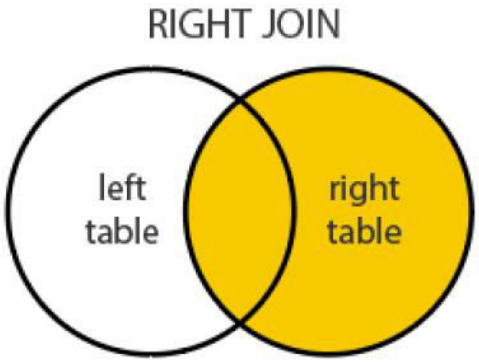


Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

```
SELECT staff.name,staff.position,dep.dno,dep.dname
  FROM staff RIGHT JOIN dep ON
  staff.dno=dep.dno
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

NAME	POSITION	DNO	DNAME
JOHN	MANAGER	D100	HR
SUSAN	MANAGER	D101	FINANCE
ANN	PROJECT MANAGER	D103	LOGISTICS
Null	Null	D104	BANKING
Null	Null	D105	MARKETING
MARY	PROJECT MANAGER	D101	HR

4. FULL OUTER JOIN

- FULL JOIN returns all matching records from both tables whether the other table matches or not.
- FULL JOIN can potentially return very large datasets.
- These two: FULL JOIN and FULL OUTER JOIN are the same.

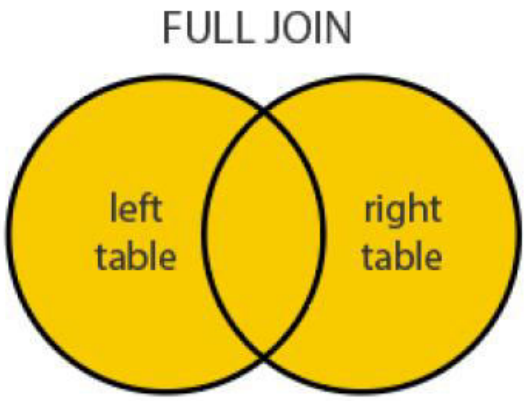


Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

```
SELECT staff.name,staff.position,dep.dno,dep.dname
  FROM staff FULL JOIN dep ON
  staff.dno=dep.dno
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Department

DNO	DNAME
D100	HR
D101	FINANCE
D103	LOGISTICS
D104	BANKING
D105	MARKETING

NAME	POSITION	DNO	DNAME
JOHN	MANAGER	D100	HR
SUSAN	MANAGER	D101	FINANCE
ANN	PROJECT MANAGER	D103	LOGISTICS
DAVID	PROJECT MANAGER	*null*	*null*
Mary	PROJECT MANAGER	D101	FINANCE
Null	Null	D104	BANKING
Null	Null	D105	MARKETING

EXAMPLES

Q1) Get the name of the staff working in HR department

Q2) Get the average salary of staffs working in Finance dept

Q3) Get the details of block in which Ann is working

Example

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Q1) Get the name of the staff working in HR department

```
SELECT staff.name FROM staff,dep WHERE staff.dno=dep.dno
   AND dep.dname='HR';
```

Example

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Q2) Get the average salary of staffs working in Finance dept

SELECT AVG(staff.salary) FROM staff,dep WHERE staff.dno=dep.dno AND dep.dname='FINANCE';

Example

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Q3) Get the details of block in which Ann is working

SELECT dep.block FROM staff,dep WHERE staff.dno=dep.dno AND staff.name='ANN';

Example

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Q4) Get the name of position working in APJ block

```
SELECT staff.position FROM staff,dep WHERE
    staff.dno=dep.dno AND dep.block='APJ BLOCK';
```

Database Management System
Module 3

Lect 6 :
-> SQL- SUB QUERY or NESTED QUERY

SUB QUERY or NESTED QUERY

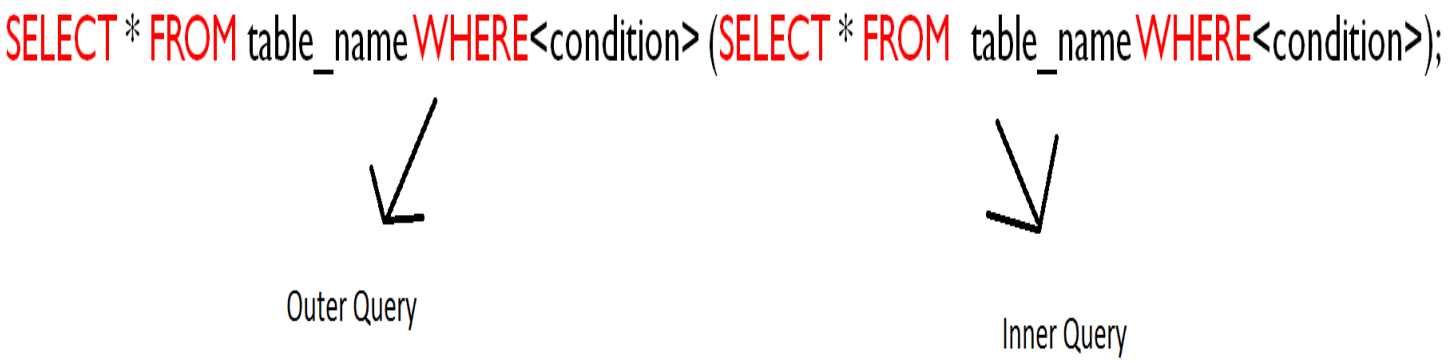
- A query inside another query is called Nested/Sub Query
- Some time to get particular information from a database you may need to fire two separate SQL queries, subQuery is a way to combine or join them in a single query.

Syntax :

```
SELECT * FROM table_name WHERE<condition> (SELECT * FROM table_name WHERE<condition>);
```

Outer Query

Inner Query



- SQL query which is on the inner part of the main query is called inner query while the outer part of the main query is called outer query.

Syntax 2 : Writing more than 2 queries

- SELECT * FROM table_name WHERE<condition>(SELECT * FROM table_name WHERE <condition>(SELECT * FROM table_name)))

Classification of SUB QUERY

- SUB QUERY is classified into two
  - Non Corelated Sub Query: -
    - Outer query is executed after executing Inner query
  - Correlated Sub Query
    - Inner query is executed after executing Outer query

NON CORELATED SUB QUERY

- Outer query is executed after executing Inner query
- Outer Query depends on the result from Inner Query
- Non Corelated Sub Query is classified into two
  - Single Row Sub Query
  - Multiple Row Sub Query

Single Row Sub Query

- When a Sub Query returns a single value , it is known as Single Row Sub Query
- It uses operators like =, > , <, >=, <=

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

- Eg : Get the employee details from the table employee having the highest salary

Single Row Sub Query

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Eg : Get the employee details from the table employee having the highest salary

Step 1: Write the inner query

```
SELECT MAX(salary) FROM staff;
```

Step 2: Write the outer query

```
SELECT * FROM staff WHERE salary = (Inner Query)
```

Step 3 : Write Sub Query = Outer query + Inner Query

```
SELECT * FROM staff WHERE salary = (SELECT MAX(salary)
    FROM staff);
```

RESULT

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100

Wrong Assumptions

- SELECT * FROM staff WHERE salary= MAX (salary) ;
- This is WRONG because an Aggregate Function cannot appear in WHERE clause

Multiple Row Sub Query

- When a Sub Query returns more than a single value , it is known as Multiple Row Sub Query
- Use the operator IN
- Cannot use the operator =,>,< .....

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

- Eg : Get the staff details whose department is same as the department of the employee 'Mary'

Multiple Row Sub Query

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

- Get the staff details whose department is same as the department of the employee 'Mary'

Step 1: Write the inner query

SELECT dno FROM staff WHERE name='Mary'

Step 2: Write the outer query

SELECT * FROM staff WHERE dno IN (Inner Query)

Step 3 : Write Sub Query = Outer query + Inner Query

SELECT * FROM staff WHERE dno IN (SELECT dno FROM employee WHERE name='Mary');

SELECT * FROM staff WHERE dno IN
(SELECT dno FROM employee WHERE name='Mary');

Result

SNO	NAME	SALARY	POSITION	DNO
SL101	SUSAN	4000	MANAGER	D101
SL104	MARY	9000	PROJECT MANAGER	D101

Non correlated sub query including multiple tables

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Get the details of employee working in any of the dept

SELECT * FROM staff WHERE dno IN (SELECT dno FROM dep);

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Get the name and position of the employee working in FINANCE dept

```
SELECT name,position FROM staff WHERE dno IN (SELECT dno FROM dep WHERE dname='FINANCE');
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

- Get the average salary of staffs working in Finance dept

SELECT AVG(salary) FROM staff WHERE dno IN (SELECT dno FROM dep WHERE dname='FINANCE');

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Get the details of block in which Ann is working

```
SELECT block FROM dep WHERE dno IN
(SELECT dno FROM staff WHERE
name='ANN');
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Get the name of position working in APJ block

SELECT position FROM staff WHERE dno IN (SELECT dno FROM dep WHERE block='APJ BLOCK'); D101

Example 2

- Eg : Retrieve ename, job, salary of the employees whose salary is less than the salary of the employee whose empno = 7876

```
mysql> select *from emp;
+-------+--------+-----------+------+---------+--------+
| empno | ename  | job       | mgr  | sal     | deptno |
+-------+--------+-----------+------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 |  800.00 |     20 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 1600.00 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 1250.00 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 2975.00 |     20 |
|  7654 | MARTIN | SALESMAN  | 7698 | 1250.00 |     30 |
|  7698 | BLAKE  | MANAGER   | 7839 | 2850.00 |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 2450.00 |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 3000.00 |     20 |
|  7839 | KING   | PRESIDENT | NULL | 5000.00 |     10 |
|  7844 | TURNER | SALESMAN  | 7698 | 1500.00 |     30 |
|  7876 | ADAMS  | CLERK     | 7788 | 1100.00 |     20 |
|  7900 | JAMES  | CLERK     | 7698 |  950.00 |     30 |
|  7902 | FORD   | ANALYST   | 7566 | 3000.00 |     20 |
|  7934 | MILER  | CLERK     | 7782 | 1300.00 |     10 |
+-------+--------+-----------+------+---------+--------+
```

Employee table

- Eg : Retrieve ename, job, salary of the employees whose salary is less than the salary of the employee whose empno = 7876

1. We will find out the salary of the employee whose empno=7876. the query is as under:

```
mysql> SELECT SAL FROM EMP WHERE EMPNO= 7876;
+----------+
| SAL      |
+----------+
| 1100.00  |
+----------+
```

2. Now in second query we will apply the condition from which we will find the ename, job and sal. We will use the query as under:

```
mysql> SELECT ENAME, JOB, SAL FROM EMP
    -> WHERE SAL < 1100;
+--------+---------+----------+
| ENAME  | JOB     | SAL      |
+--------+---------+----------+
| SMITH  | CLERK   | 800.00   |
| JAMES  | CLERK   | 950.00   |
+--------+---------+----------+
```

- The above two queries can be used as single query by using the concept of subquery. It will combine the result into a single query as under:

```
mysql> SELECT ENAME, JOB, SAL FROM EMP
    -> WHERE SAL< (SELECT SAL FROM EMP WHERE EMPNO=7876);
+-------+-------+--------+
| ENAME | JOB   | SAL    |
+-------+-------+--------+
| SMITH | CLERK | 800.00 |
| JAMES | CLERK | 950.00 |
+-------+-------+--------+
```

Eg: Retrieve ename, job, salary of the employees whose job is same as that of ‘miller’

```
mysql> select *from emp;
+-------+--------+-----------+------+---------+--------+
| empno | ename  | job       | mgr  | sal     | deptno |
+-------+--------+-----------+------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 |  800.00 |     20 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 1600.00 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 1250.00 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 2975.00 |     20 |
|  7654 | MARTIN | SALESMAN  | 7698 | 1250.00 |     30 |
|  7698 | BLAKE  | MANAGER   | 7839 | 2850.00 |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 2450.00 |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 3000.00 |     20 |
|  7839 | KING   | PRESIDENT | NULL | 5000.00 |     10 |
|  7844 | TURNER | SALESMAN  | 7698 | 1500.00 |     30 |
|  7876 | ADAMS  | CLERK     | 7788 | 1100.00 |     20 |
|  7900 | JAMES  | CLERK     | 7698 |  950.00 |     30 |
|  7902 | FORD   | ANALYST   | 7566 | 3000.00 |     20 |
|  7934 | MILER  | CLERK     | 7782 | 1300.00 |     10 |
+-------+--------+-----------+------+---------+--------+
```

Employee table

SELECT ename, job,sal FROM employee WHERE job IN (SELECT job FROM employee WHERE ename=‘miller’);

SELECT ename, job,sal FROM employee WHERE job IN (SELECT job FROM employee WHERE ename='miller');

Result

Ename	Job	Sal
Smith	Clerk	800
Adam	Clerk	1100
James	Clerk	950
Miller	Clerk	1300

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

- SELECT fname from staff where salary=(select max(salary) from staff);
- SELECT fname from staff where salary IN (select salary from staff where fname='Ann');
- 4. SELECT position from staff where=salary=(select max(salary) from staff);
- 5.

eno	ename	dno
e100	Sooraj	d100
e101	Deepu	d101
e102	vivek	d102

dno	dname	salary
d100	CSE	25000
d101	EEE	22000
d102	Cyb	25000
d103	AO	25000
d104	AE	35000
d105	RA	35000

Qstn : select department name that does not have even a single employee

```
SELECT d.dname FROM dept d WHERE NOT EXISTS (SELECT e.ename FROM emp e WHERE e.dno=d.dno);
```

dname
AO
AE
RA

Qstn : select department name that have atleast one employee

```
SELECT d.dname FROM dept d WHERE EXISTS (SELECT e.ename FROM emp e WHERE e.dno=d.dno);
```

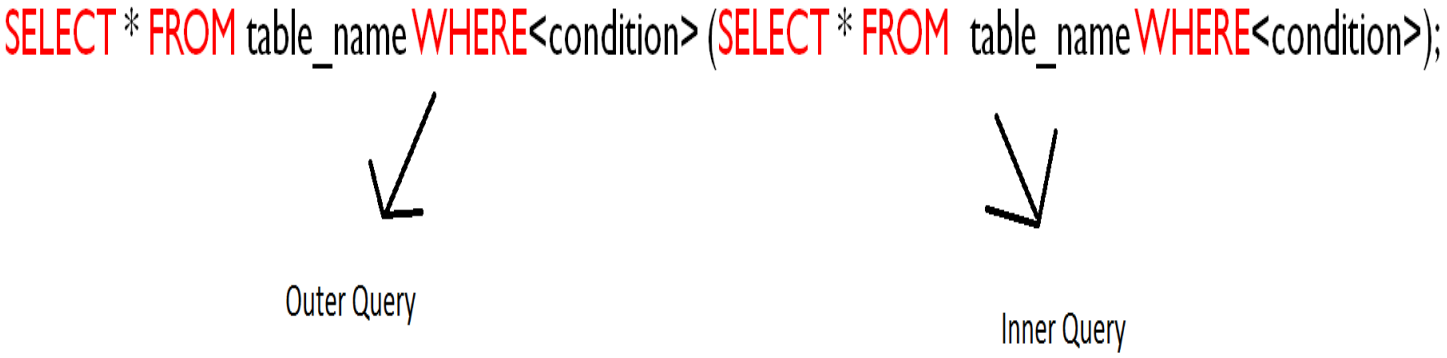
dname
CSE
EEE
Cyb

Database Management System
Module 3

Lect 7 :
CORRELATED SUB QUERY

Correlated Subquery

- It is a subquery that uses values from outer query.
- Inner query is executed after executing Outer query



Correlated Query Example

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Get the details of employee working in any of the dept

```
SELECT * FROM staff WHERE EXISTS (SELECT * FROM dep
  WHERE staff.dno=dep.dno);
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

- Get the name and position of the employee working in FINANCE dept

```
SELECT name,position FROM staff WHERE
  EXISTS(SELECT * FROM dep WHERE
  staff.dno=dep.dno AND dname='FINANCE');
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

- Get the average salary of staffs working in Finance dept

```
SELECT AVG(SALARY) FROM STAFF WHERE
  EXISTS (SELECT *  FROM DEP WHERE
  staff.dno=dep.dno AND dname='FINANCE');
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Get the details of block in which Ann is working

```
SELECT block FROM dep where
  EXISTS(SELECT * FROM staff WHERE
  staff.dno=dep.dno AND name='ANN');
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

Get the name of position working in APJ block

```
SELECT position FROM staff WHERE
  EXISTS(SELECT * FROM dep WHERE
  staff.dno=dep.dno AND block='APJ
  BLOCK');
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

select department name that does not have even a single employee

```
SELECT dname FROM dep WHERE NOT EXISTS (SELECT * FROM staff WHERE staff.dno=dep.dno);
```

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : DEP

DNO	DNAME	BLOCK
D100	HR	MG BLOCK
D103	LOGISTICS	CVR BLOCK
D104	BANKING	VKN BLOCK
D105	MARKETING	PJN BLOCK
D101	FINANCE	APJ BLOCK

select department name that have atleast one staff

```
SELECT dname FROM dep WHERE EXISTS
(SELECT * FROM staff WHERE
staff.dno=dep.dno);
```

Views in SQL

- A view in SQL terminology is a single table that is derived from other tables
- These other tables can be base tables or previously defined views.
- A VIEW does not require any storage in a database because it does not exist physically.

Create a SQL View

- Syntax
- CREATE VIEW Name AS SELECT column1, Column2...Column N FROM tables WHERE conditions

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

```
CREATE VIEW staff_sal AS SELECT
name,position FROM staff WHERE salary>5000;
```

Name	Position
JOHN	MANAGER
DAVID	PROJECT MANAGER
MARY	PROJECT MANAGER

View :staff_sal

Display view

SELECT * FROM staff_sal;

Name	Position
JOHN	MANAGER
DAVID	PROJECT MANAGER
MARY	PROJECT MANAGER
View :staff_sal	

SELECT name FROM staff_sal ;

Name
JOHN
DAVID
MARY
View :staff_sal

UPDATE VIEW

- UPDATE < view_name > SET<column>=<value 1>,<column2=value2> .....WHERE <condition>;
- Eg : UPDATE staf_sal SET position='MANAGER' WHERE name='DAVID';

Name	Position
JOHN	MANAGER
DAVID	MANAGER
MARY	PROJECT MANAGER

View :staff_sal

DELETE VIEW

- SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

Syntax

DROP VIEW view_name;

Eg:

DROP VIEW staff_sal;

Exercise

- 1. Get the Name of the student studying in any of the department

Query : SELECT name FROM student WHERE EXISTS (SELECT * FROM dept WHERE student.roll=dept.roll);

- 2. What is the department which does not have any student

SELECT dname FROM dept WHERE not EXISTS (SELECT * FROM student WHERE student.roll=dept.roll);

- 3. Get the Name of the student studying in CSE dept

```
select name from student where exists(select * from dept where
    student.roll=dept.roll and dname='CSE');
```

- 4. Get the department name where Anand is studying

```
select dname from dept where exists(select * from student where
    student.roll=dept.roll and name='Swetha');
```

- 5. Create a view stud_name which has the name of the students having age above 25

```
Create view stud_name AS SELECT name FROM student WHERE
    age>'25'
```

Database Management System
Module 3

Lect 8 :
SQL ASSERTIONS & TRIGGERS

ASSERTIONS

- An assertion is a predicate expressing a condition we wish the database to always satisfy
- Assertions are used to specify integrity constraint that cannot be implemented using constraints like Primary Key, Not Null,Unique, Default,Check etc

- Example : salary of Manager in staff table must not be less than Project Manager

Syntax

```
CREATE ASSERTION <assertion-name>
CHECK <predicate>;
```

```
DROP ASSERTION <assertion-name>
```

Example

Table : Staff

SNO	NAME	SALARY	POSITION	DNO
SL100	JOHN	30000	MANAGER	D100
SL101	SUSAN	4000	MANAGER	D101
SL102	DAVID	12000	PROJECT MANAGER	D102
SL103	ANN	5000	PROJECT MANAGER	D103
SL104	MARY	9000	PROJECT MANAGER	D101

Example : salary of Manager in staff table must not be less than Project Manager

```
CREATE ASSERTION salary_staff
CHECK( NOT EXISTS
(SELECT * FROM staff  WHERE
  position='MANAGER' AND salary<(SELECT
  salary FROM staff WHERE position ='Project
  Manager')));
```

Example 2

Book id	B_name	Price	Catergory

Eg 2 Price of the textbook category is not less than minimum price of novel

CREATE ASSERTIONS price_constraint

CHECK (NOT EXISTS (SELECT * FROM book WHERE category='Text book' AND price < (SELECT price FROM book WHERE category='novel')))

NOT EXIST – indicates that the result of the query must be empty

Assertion is violated if the result of query inside NOT EXISTS clause is not empty

Table : Employee		
eid	Name	Salary

Table : attendance		
eid	Punching _In	Punching Out

- Create assertion to check whether any staff has punched in after 10:00 AM

```
CREATE ASSERTION late_punch
CHECK (NOT EXISTS
(SELECT name FROM employee WHERE eid
  IN(SELECT  eid FROM attendance WHERE
  employee.eid=attendance.eid AND
  attendence.punching_in>10)));
```

TRIGGERS

- Triggers are SQL codes that are automatically excuted in response to a certain events on a particular table.
- A trigger is a statement that is automatically executed by the system as a side effect of a modification to the database.
- We need to
  - ► Specify the conditions under which the trigger is executed.
  - ► Specify the actions to be taken by the trigger.

Three modules for Trigger

1. Event – eg Insert/update etc
2. Condition – is evaluated
3. Action – takes place based on the condition

Syntax

CREATE TRIGGER <trigger name>

(AFTER|BEFORE) <trigger event> ON
<table name>

[FOR EACH ROW]

[WHEN <condition>]

<trigger action>

Example

- Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.
- Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used

Query

```
CREATE TRIGGER stud_marks
BEFORE  INSERT on Student
FOR EACH ROW
set Student.total = Student.subj1  + Student.subj2 +
   Student.subj3,
Student.per = Student.total * 60 / 100
```

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0,
Query OK, 1 row affected (0.09 sec)

mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
```

Database Management System
Module 3

Lect 9 :
PHYSICAL DATA ORGANIZATION

Physical Files

- Physical files contain the actual data that is stored on the system, and a description of how data is to be stored
- They contain only one record format.

Logical files

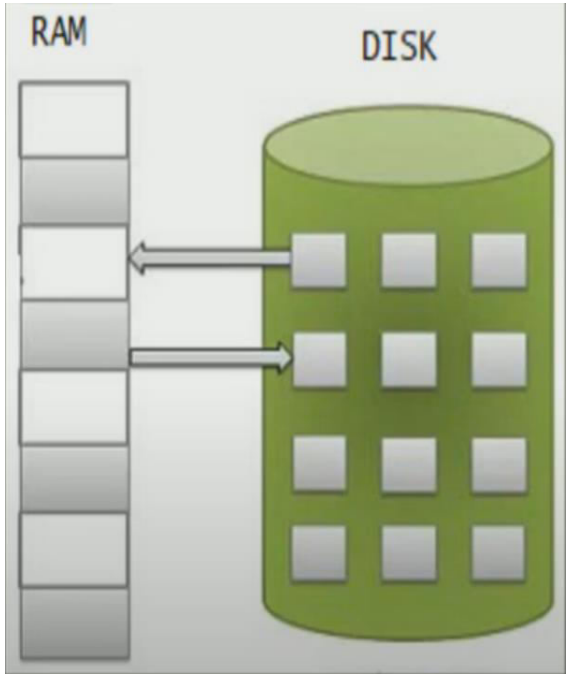
- Logical files do not contain data.
- They contain a description of records found in one or more physical files.
- A logical file is a view or representation of one or more physical files.
- Logical files that contain more than one format are referred to as multi-format logical files.

DATA ORGANIZATION & ACCESS METHODS

- Data organization refers to the organization of the data of a file into records, blocks, and access structures;
- Access method, on the other hand, provides a group of operations that can be applied to easily access a file.

PHYSICAL DATA ORGANIZATION

- Data is stored on the form of files
- Disk data is stored in blocks
- In order to read a data which is stored in the block, the entire block is read and copied to RAM
- Blocks can be
  - Fixed sized blocks
  - Variable sized blocks



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

- A file is a sequence of records.
- All records in a file are of the same record type.
- If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records.
- If different records in the file have different sizes, the file is said to be made up of variable-length records.

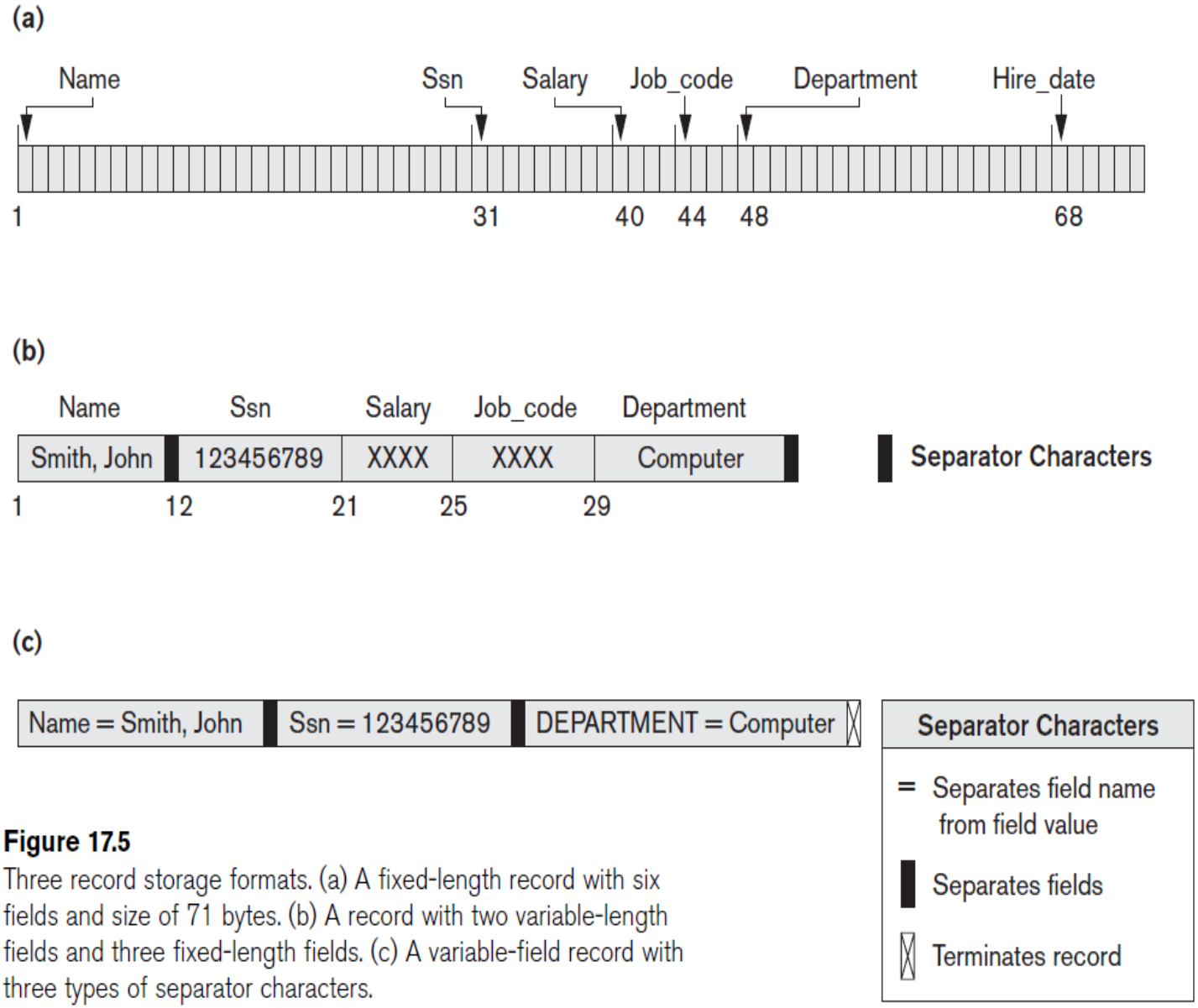
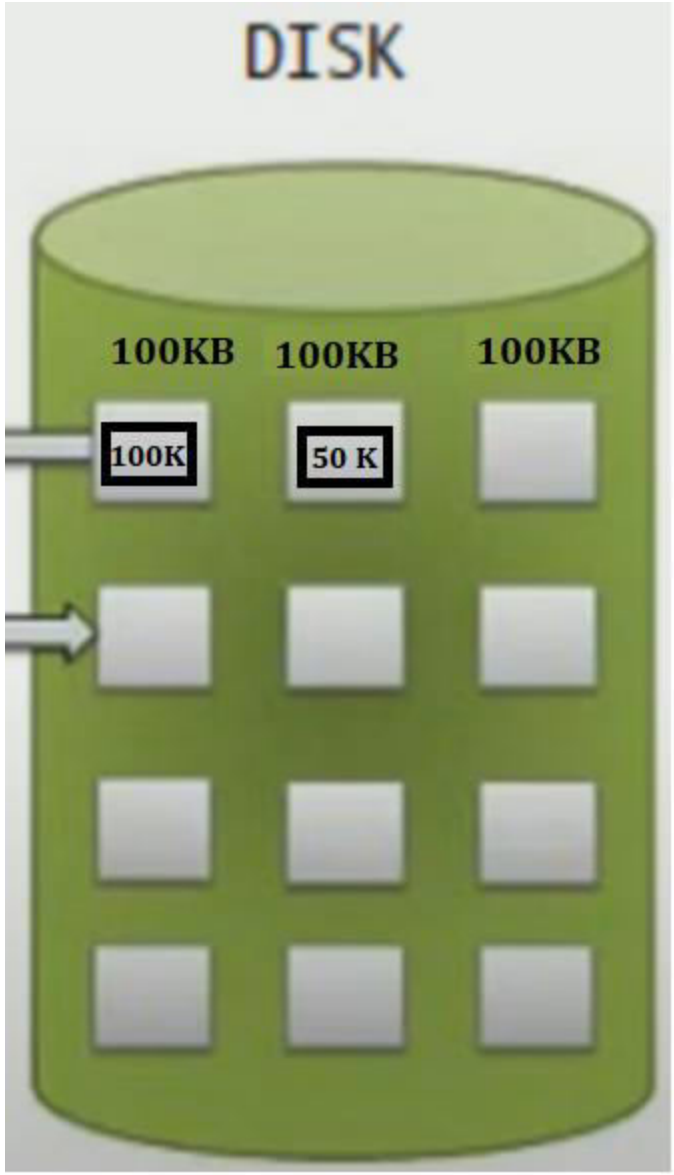


Figure 17.5
Three record storage formats. (a) A fixed-length record with six fields and size of 71 bytes. (b) A record with two variable-length fields and three fixed-length fields. (c) A variable-field record with three types of separator characters.

- Figure (a) represent fixed-length EMPLOYEE record having record size of 71 bytes
- Figure (b) represent variable-length EMPLOYEE record, here we do not know the exact length of some field values. To determine the bytes within a particular record that represent each field, we can use special **separator characters**
- Figure (c) represent variable-length EMPLOYEE record, here number of fields that actually appear in a typical record is less, we can include in each record a sequence of <field-name, field-value> pairs rather than just the field values

Spanned & Unspanned Organization

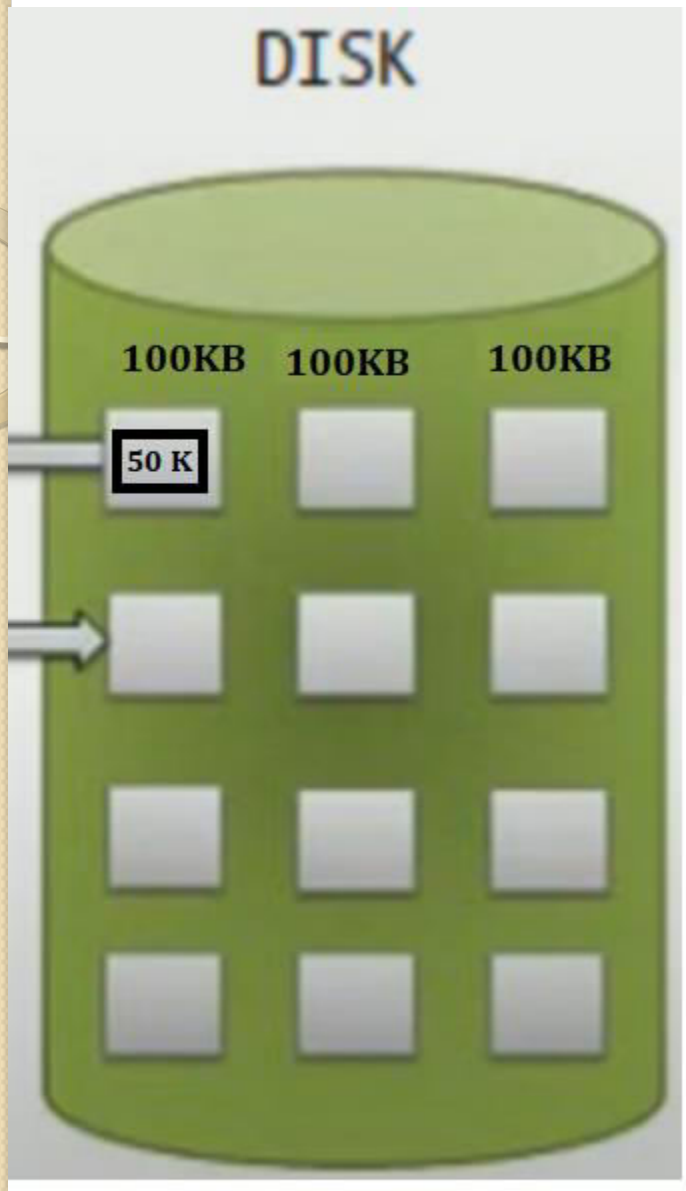
- The records of a file must be allocated to disk blocks
- When the block size is larger than the record size, each block will contain numerous records.
- If a record is stored in single block – Unspanned Organization
- Some files may have unusually large records that cannot fit in one block.
- Part of the record can be stored on one block and the rest on another, such organization is called Spanned Organization



Record (150 KB)

Student Table			
Roll	**Name**	**Dept**	**Age**
1	Akhil	CS	21
2	Bipin	CS	20
3	Deepak	CS	21

SPANNED ORGANIZATION



Record (50 KB)

Student Table			
Roll	**Name**	**Dept**	**Age**
1	Akhil	CS	21
2	Bipin	CS	20
3	Deepak	CS	21

UNSPANNED ORGANIZATION

Blocking Factor(bfr)

- Bfr is the no. of records available per block
- Suppose that the block size is *B* bytes.
- For a file of fixed-length records of size *R* bytes, with $B \geq R$,
- Then no. of records per block /blocking factor

  $bfr = \lfloor B/R \rfloor$

where the $\lfloor (x) \rfloor$ (floor function) rounds down the number x to an integer.

- R may not divide B exactly, so we have some unused space in each block equal to

  $B - (bfr * R)$ bytes

- We can use bfr to calculate the number of blocks b needed for a file of r records:

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

where the $\lceil (x) \rceil$ (ceiling function) rounds the value x up to the next integer

for Spanned organization

number of blocks

$= \lceil$ no. of records X recordsize /blocksize $\rceil$

Example 1

- Number of records=1000
- record size=100 bytes
- block size=512 byte

- for Spanned organization

number of blocks needed= ⌈no. of records X recordsize ÷ blocksize ⌉

⌈1000 × 100 ÷ 512 ⌉ = ⌈195.31 ⌉ = 196 blocks

- Unspanned organization

Bfr = No of record per block= ⌊ blocksize /record size ⌋ = ⌊512/100⌋
= ⌊ 5.12 ⌋ = 5 record/block

Total no of blocks= ⌈ No. of records/ bfr ⌉⌈ 1000/5 ⌉ = 200 blocks

Example 2

- How many disk block required to store 2000 records each of size 100bytes. Given block size is 512 bytes

- For Spanned organisation

number of blocks needed= ⌈no. of records X recordsize ÷ blocksize ⌉

= 2000 × 100 ÷ 512 = ⌈390.6⌉ =391 block

- For Unspanned organisation

Bfr/ No. of record per block= ⌊ blocksize /record size ⌋

= ⌊512/100⌋ = 5 records per block

Total no of blocks= ⌈ No. of records/ bfr ⌉ = 2000/5 =400

PINNED AND UN PINNED RECORDS

- A record is said to be pinned down or pinned record if there exists a pointer to it somewhere in the database.
- For example, inorder to locate a record, the table contains a pointer to the record and the record becomes pinned down. The pinned records cannot be moved without reason because in that case the pointers pointing to these records will suspend.
- A record is said to be unpinned record, if there does not exist any pointer pointing to it in the database. In fact, it is the independent record.

Database Management System
Module 3

Lect 10 :
INDEX STRUCTURES

INDEX STRUCTURES

- The index structures are additional files on disk that provide an alternative ways to access the records without affecting the physical placement of records in the primary data file on disk.
- Access to records based on the indexing fields that are used to construct the index
- Any field /attribute of the table can be used to create an index
- A variety of indexes are possible; each of them uses a particular data structures to speed up the search

INDEX FILE

Block Anchor(Primary Key value)	Block Pointer
1	
4	
9	
.	
.	
.	
998	

DATA FILE

Sl No	Name	Marks	
1	Ram	43	Block 1
2	Suni	42	
3	Rajesh	37	
4	Jibin	32	Block 2
5	Jose	33	
6	Pradeep	35	
7	Rakhi	38	Block 3
8	Jacob	45	
9	George	43	
	.		
	.		
	.		
998	Suresh	45	Block N
999	Yadev	34	
1000	Jaideep	34	

TYPES OF INDEXING

- Single level ordered indexes
  - Primary index
  - Clustering index
  - Secondary index
- Multilevel index
- Dynamic Multilevel Indexes Using B-Trees

PRIMARY INDEX

- Primary index is applied on an ordered file of records.
- A primary index is an ordered file with fixed length records having **two fields**.
  - The first field -**primary key**—of the data file (also called anchor record),
  - Second field is a **pointer to a disk block** (Block Pointer).
- There is one index entry (or index record) in the index file for each block in the data file.
- Total number of entries in the index = number of disk blocks in the ordered data files
- Data in the index file is also divided into blocks

- **A primary index is a nondense (sparse) index, since it includes an entry for each disk block.**
- To retrieve a record, given the value K of its primary key field, a binary search is performed on the index file to find the appropriate index entry, and then retrieve the data file block

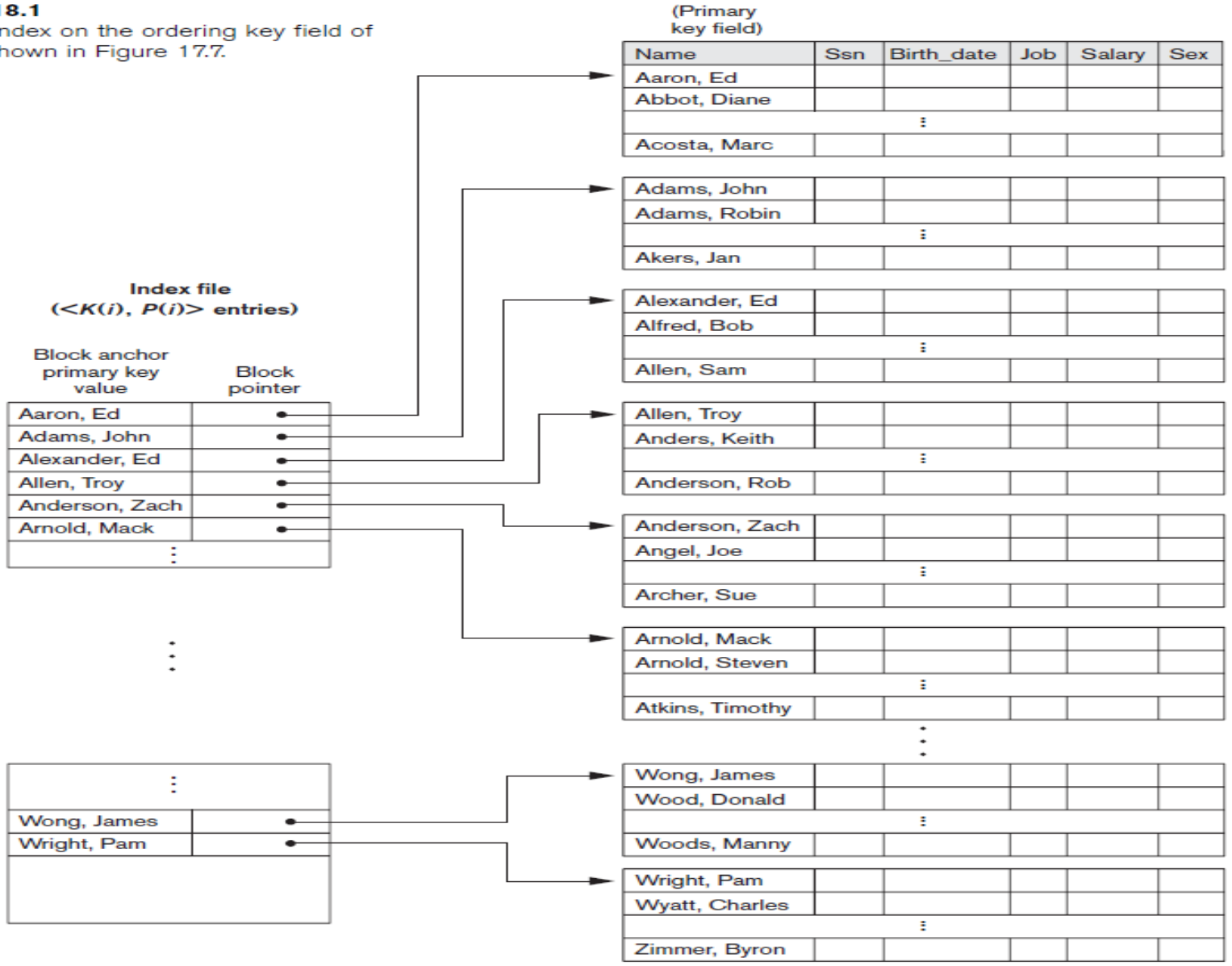
INDEX FILE

Block Anchor(Primary Key value)	Block Pointer
1	
4	
9	
.	
.	
.	
998	

DATA FILE

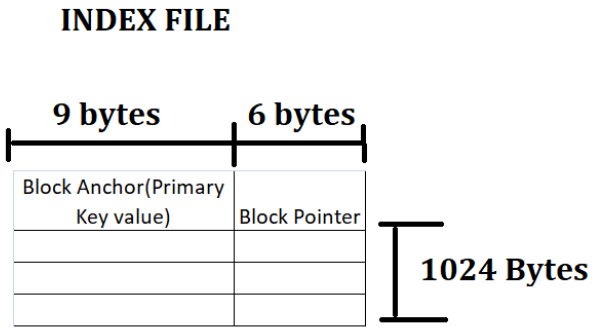
Sl No	Name	Marks	
1	Ram	43	Block 1
2	Suni	42	
3	Rajesh	37	
4	Jibin	32	Block 2
5	Jose	33	
6	Pradeep	35	
7	Rakhi	38	Block 3
8	Jacob	45	
9	George	43	
	.		
	.		
	.		
998	Suresh	45	Block N
999	Yadev	34	
1000	Jaideep	34	

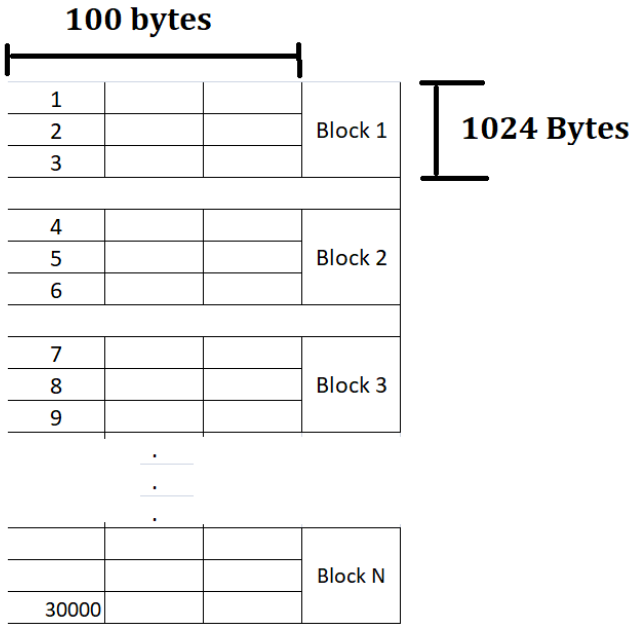
Figure 18.1
Primary index on the ordering key field of the file shown in Figure 17.7.



Numerical Problem on Primary Index

- Suppose we have an ordered file with 30,000 records and stored on a disk of block size 1024 bytes and records are of fixed size, unspanned organization. Record length = 100 bytes.
- Find Blocking factor,
- No. of blocks needed for the file,
- How many block access if using a primary index file, with an ordering key field of the file 9 bytes and block pointer size 6 bytes.





For Data File

- Blocking Factor (Bfr) = ⌊ blocksize /record size ⌋ No of record per block=

⌊ 1024 /100 ⌋ = ⌊ 10.24 ⌋ = 10 records

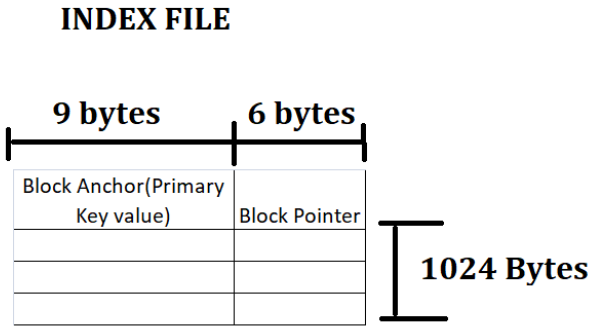
- No. of Blocks needed= no of records/bfr

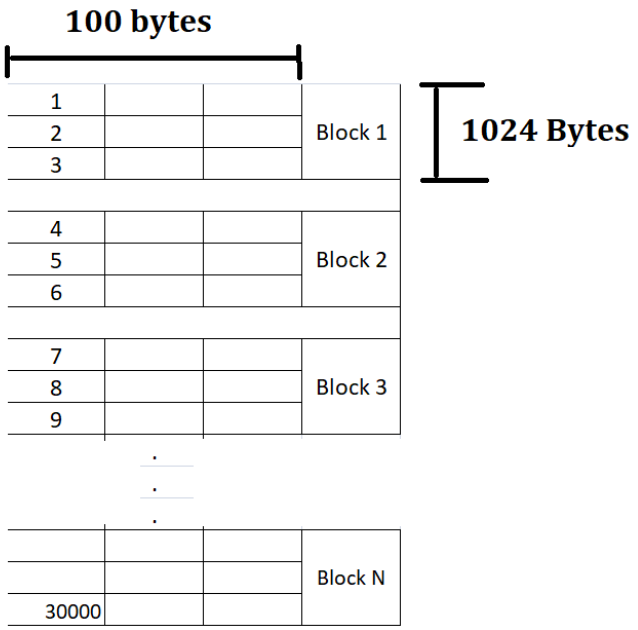
= 30000/10 = 3000 blocks

- block access required = log2 B

B-> No of blocks

= log 2 3000 =





For Index File

- Blocking Factor (Bfr) = No of record per block= $\lfloor$ blocksize /record size $\rfloor$

= $\lfloor$ 1024/15 $\rfloor$ = $\lfloor$ 68.26 $\rfloor$ = 68

- No. of Blocks needed= $\lceil$ no of records/bfr $\rceil$ =

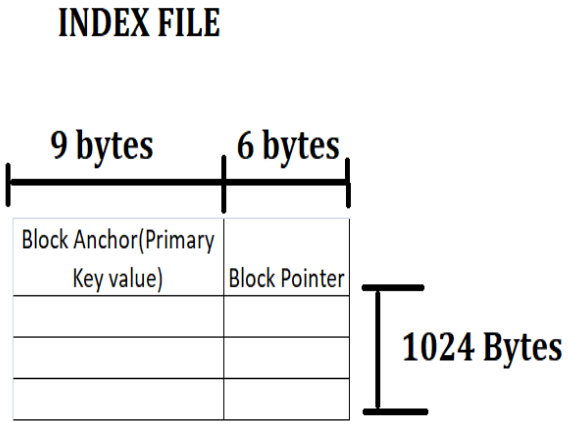
No of records in primary index = no. of blocks in data file = 3000

So No. of Blocks needed = $\lceil$ 3000/68 $\rceil$ = 45

- Block access required = log2 B = log2 45 = 6 blocks

To search for a record using index we need additional one

block access =6+1=7 block access



- **A Primary Index file contains key field of 6 bytes and block pointer size of 4 bytes .The block size is 100 bytes. The data file is stored in 160 blocks How many block access is required to access the index file**

Ans : Record size of Index file = 4+6=10 bytes

BFR = blocksize/record size = 100/10= 10 records per block

No of records in primary index = no. of blocks in data file = 100

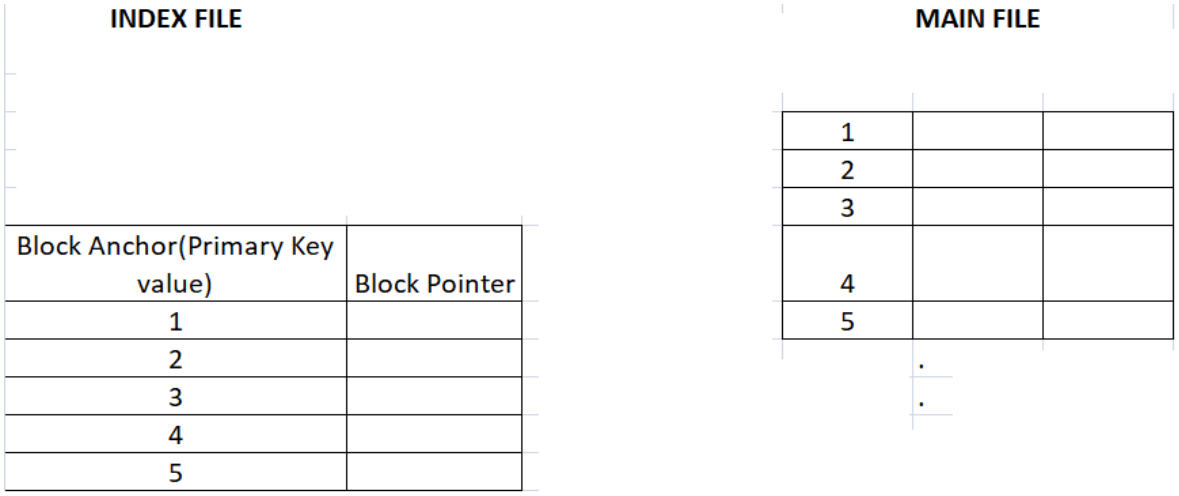
So No. of Blocks needed = $\lceil 160/10 \rceil = 16$

No of block access = log2 B = log2 16 = 4

To search for a record using index we need additional one block access =4+1=5 block access

DENSE & SPARSE INDEX

- Indexes can also be characterized as dense or sparse.
- A **dense index** has an index entry for every search key value (and hence every record) in the data file.



A **sparse** (or nondense) **index**, on the other hand, has index entries for only some of the search values.

INDEX FILE

Block Anchor(Primary Key value)	Block Pointer
1	
4	
7	
10	

MAIN FILE

1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Database Management System
Module 3

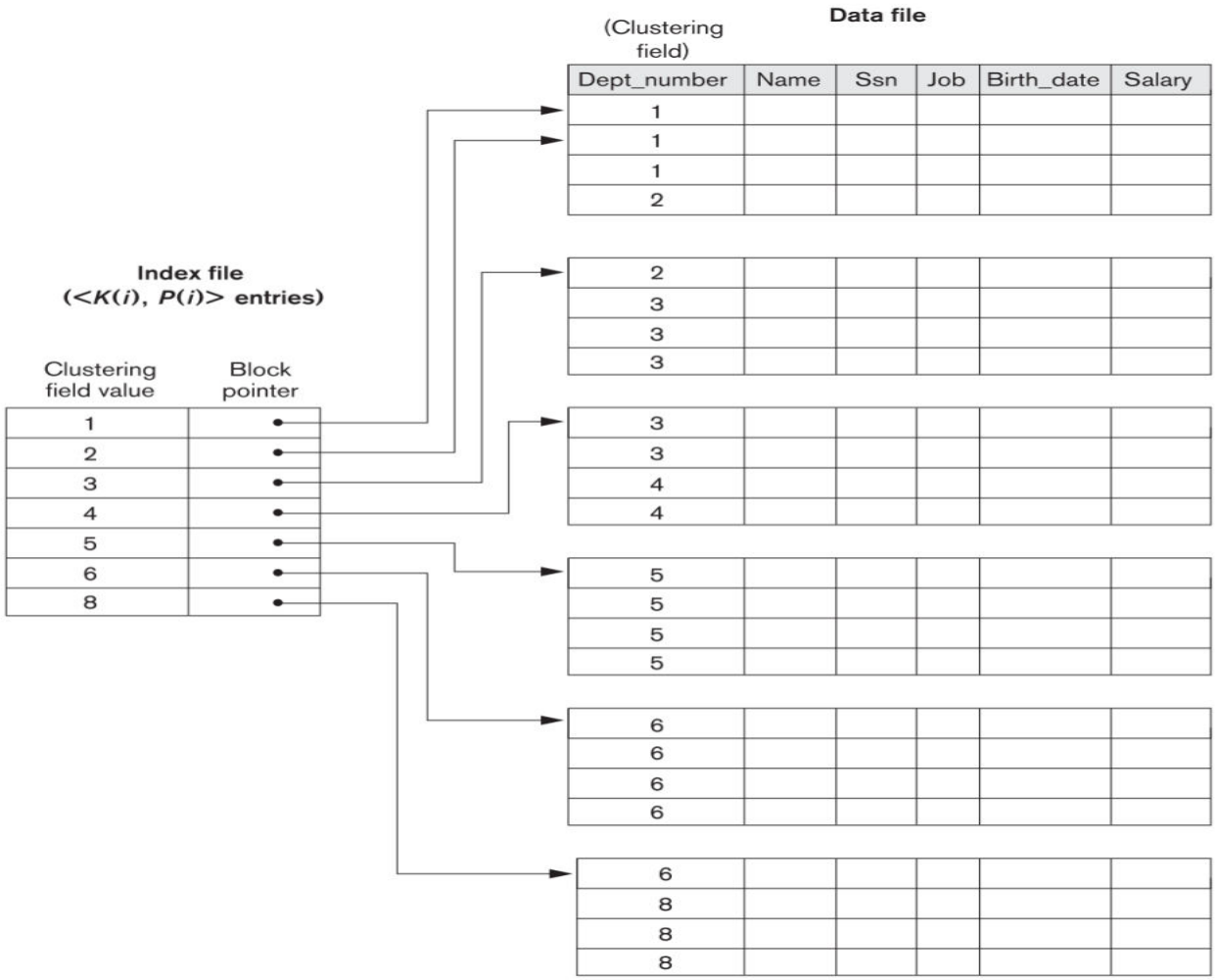
Lect 11 :
INDEX STRUCTURES Contn.
- Clustering Index
- Secondary Index

CLUSTERING INDEX

- File records are physically ordered on a **nonkey field**—(does not have a distinct value for each record)—
- such field is called the **Clustering field**.
- It is an ordered file with two fields
  - First field is the **clustering field of the data file**
  - Second field is **the block pointer**
- There is one entry in the clustering index for each of the distinct value of the clustering field.
- It is a **Non dense Index/Sparse Index**

- An index created on clustering field called a clustering index, to speed up retrieval of records that have the same value for the clustering field.

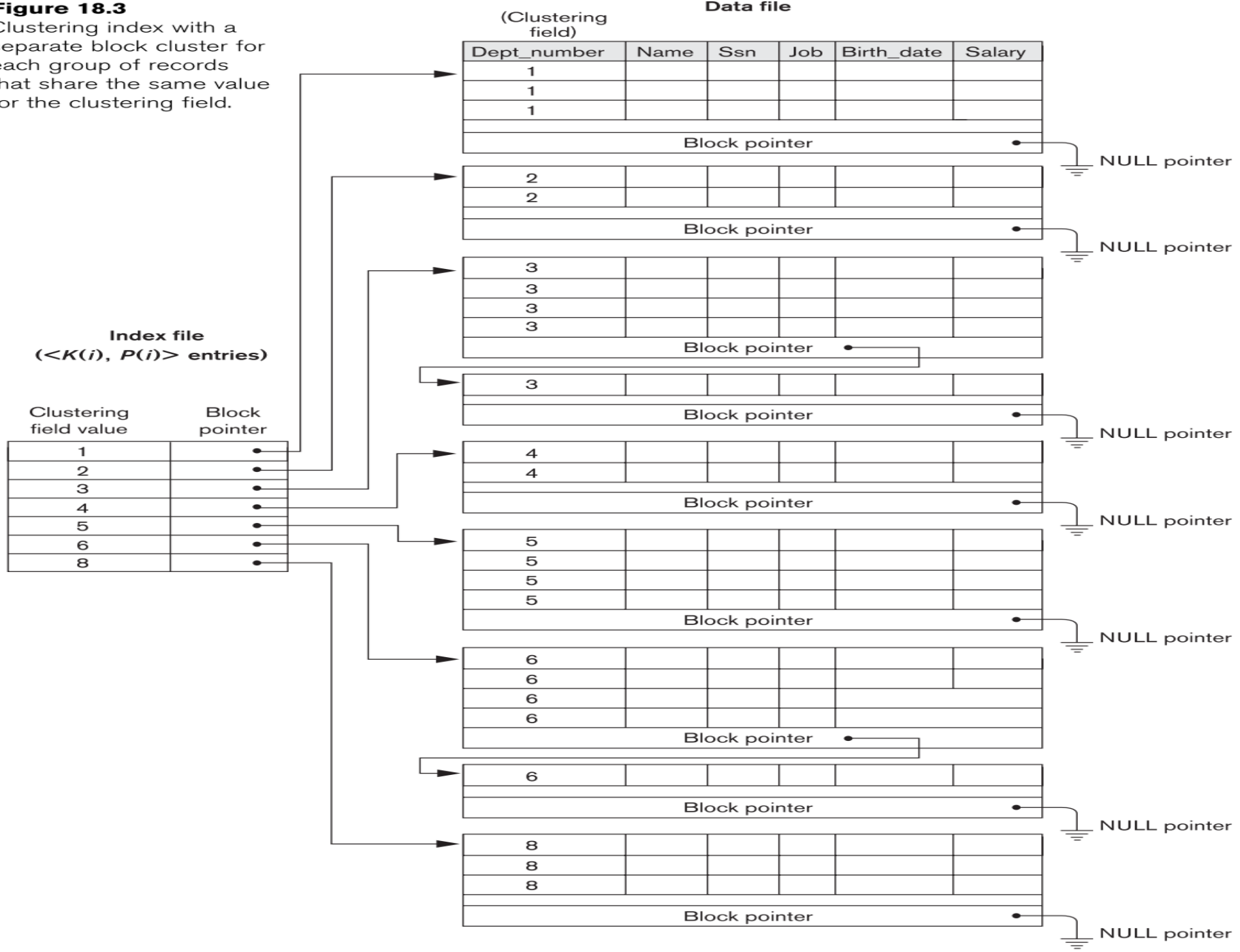
CLUSTERING INDEX



CLUSTERING INDEX

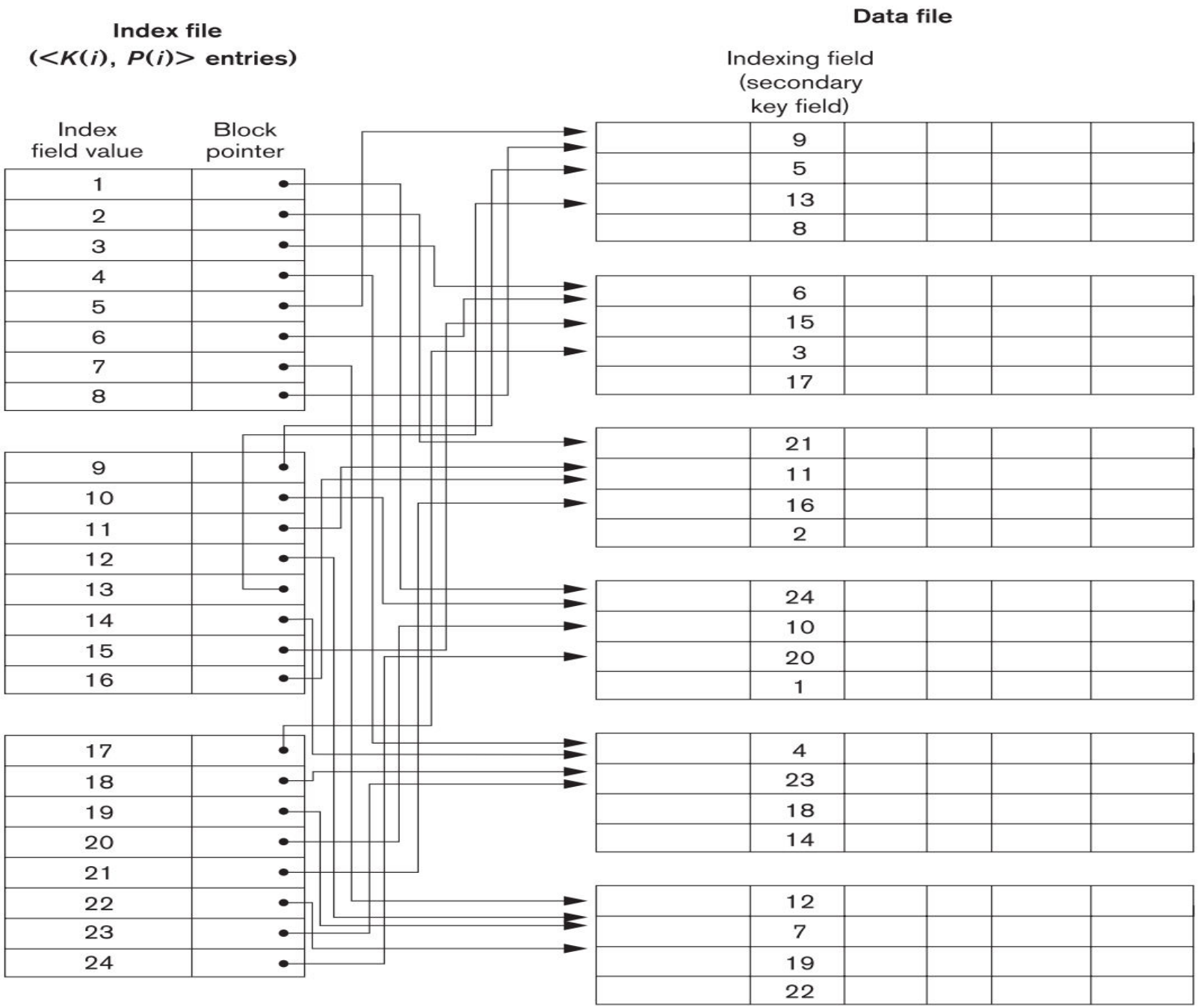
- A clustering index is another example of a **nondense index**, because it has an entry for every distinct value of the indexing field
- Here record insertion and deletion still cause problems, because the data records are physically ordered.
- To avoid the problem of insertion, it is common to **reserve a whole block for each value of the clustering field;** all records with that value are placed in the block. This makes insertion and deletion relatively straightforward.

Figure 18.3
Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.

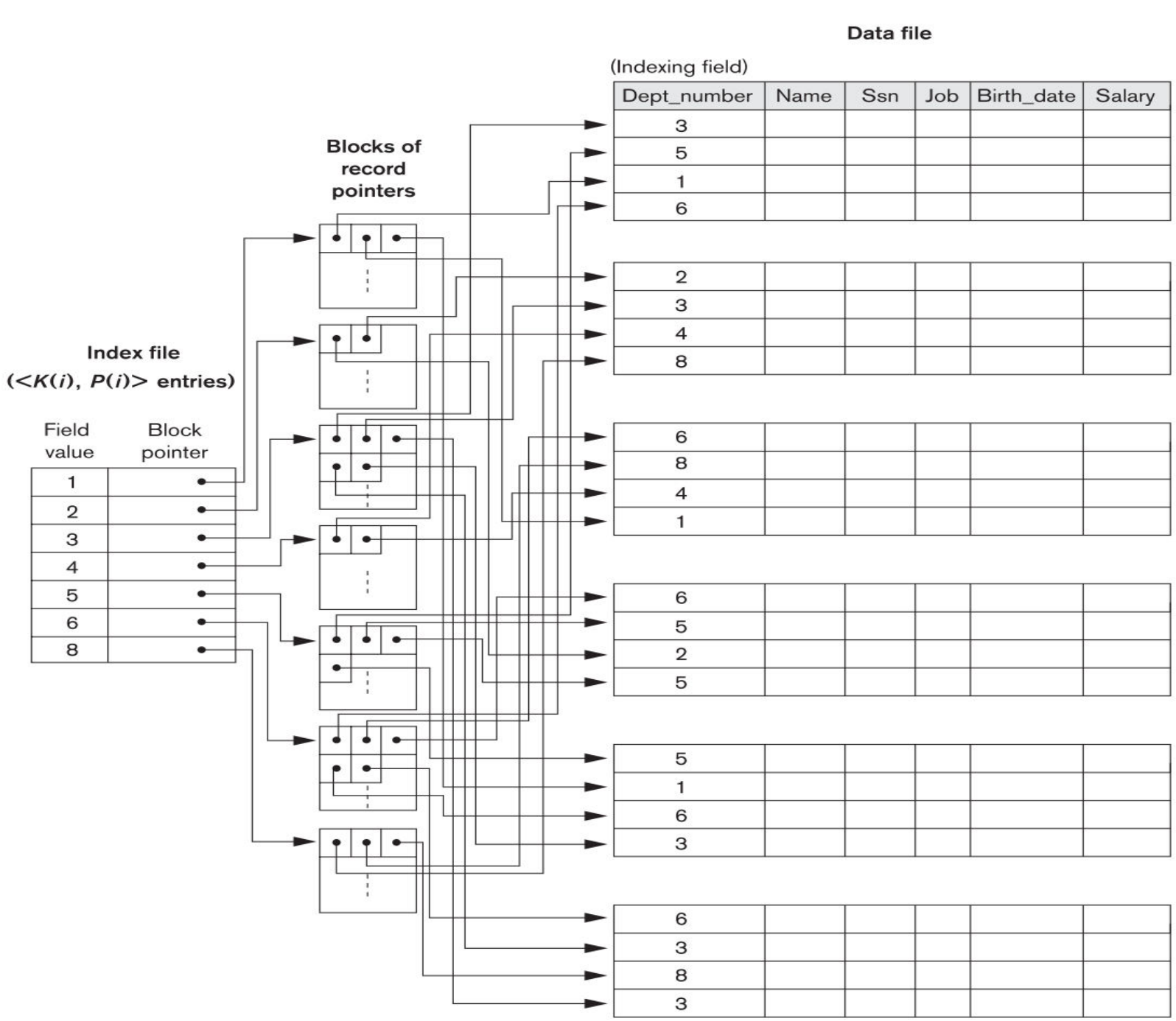


SECONDARY INDEX

- Here data file is unsorted
- Secondary index may be created on a field that is a
  - **candidate key** and has a unique value in every record,
  - **or on a nonkey field with duplicate values**.
- Index file is an ordered file with two fields
  - The first field is a key or non key
  - Second field is either a block pointer or record pointer
- It is an example for a dense index

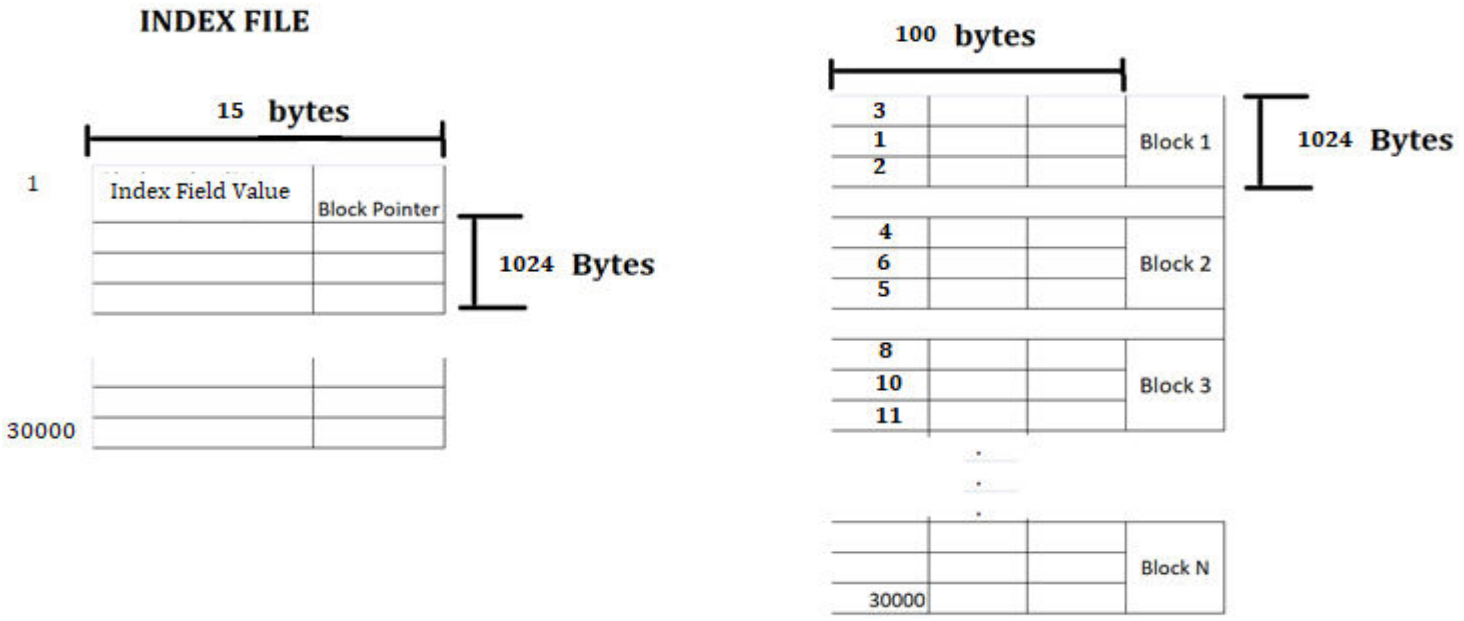

Index file
(<K(i), P(i)> entries)
Index field value
Block pointer
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Data file
Indexing field (secondary key field)
9
5
13
8
6
15
3
17
21
11
16
2
24
10
20
1
4
23
18
14
12
7
19
22

- A secondary index can also be created on non key field.
- Here there will be numerous records in the data file that have the same value for indexing field.
- the block pointer in index entry points to a disk block, which contains a set of record pointers;
- each record pointer in that disk block points to one of the data file records


Data file
(Indexing field)
Dept_number
Name
Ssn
Job
Birth_date
Salary
Blocks of record pointers
Index file
(<K(i), P(i)> entries)
Field value
Block pointer
1
2
3
4
5
6
8
3
5
1
6
2
3
4
8
6
8
4
1
6
5
2
5
5
1
6
3
6
3
8
3

Numerical Problem on Secondary Index

- Given an unordered file no. of records in the data file is 30000, block size is 1024 bytes, length of each record in data file is is 100 byte and length of each record in index file is 15 bytes. Find blocking factor, no.of blocks in the index file, no of block access required in the index file



- Blocking factor of index file =

No of record per block=

└ blocksize /record size ┘

= └ 1024/15 ┘ = └ 68.2 ┘ = 68 records

- No. of blocks in index file =

⌈ No. of records in index file / blocking factor ⌉

Since it is dense index , No. of records in index file = no. of records in data file

= ⌈ 30000/68 ⌉ = ⌈ 441.17 ⌉ = 442

- No of block access = ⌈ log 2 B ⌉ + 1

= log 2 442 =

Database Management System
Module 3

Lect 12 :
INDEX STRUCTURES Contn.
- Multilevel Index

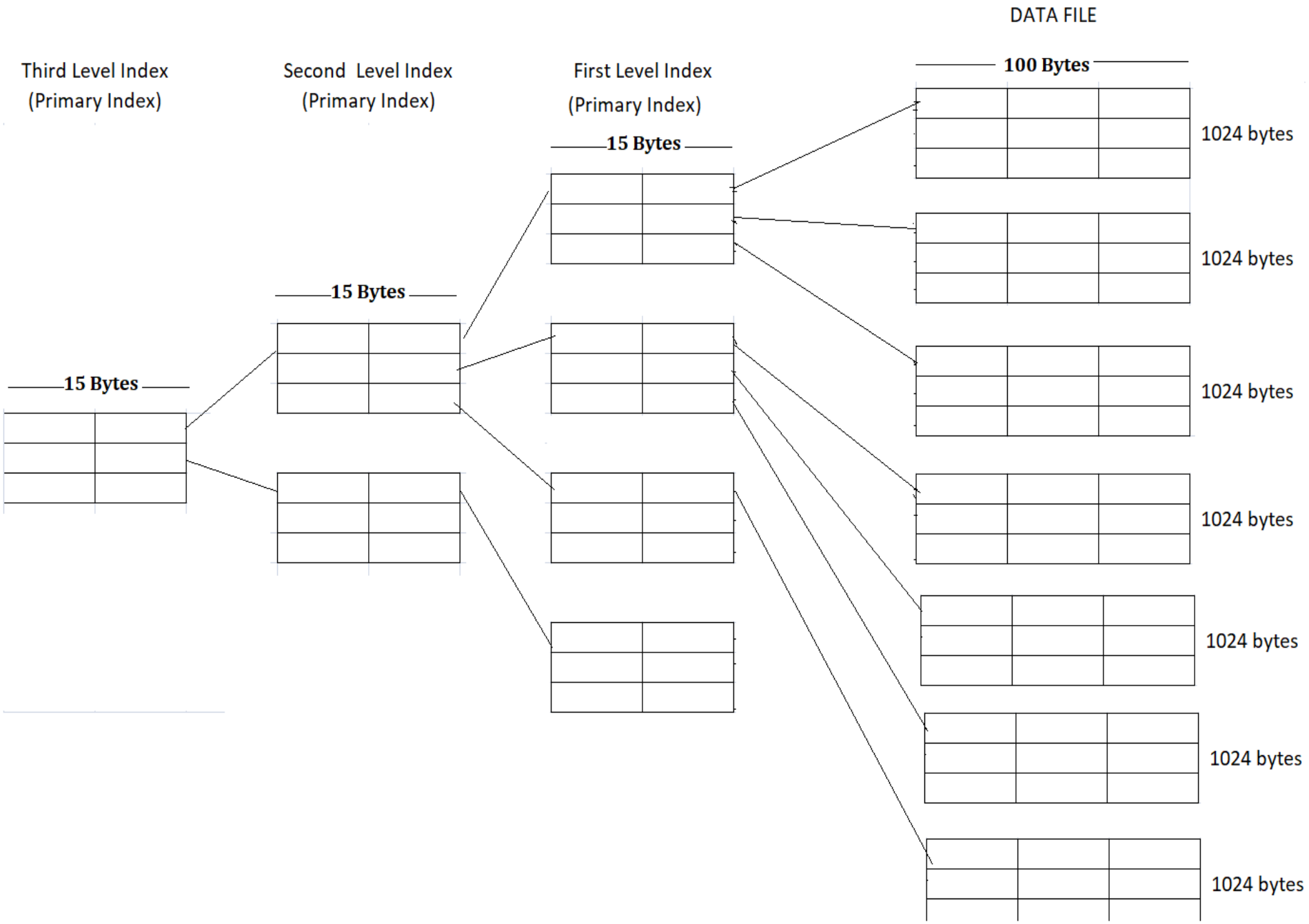
Multilevel Indexes

- A multilevel index considers the index file, as the first (or base) level of a multilevel index
- We can create a primary index for the first level; this index to the first level is called the second level of the multilevel index.
- Second level is a primary index, i.e second level has one entry for each block of the first level. This process is repeated for the second level.

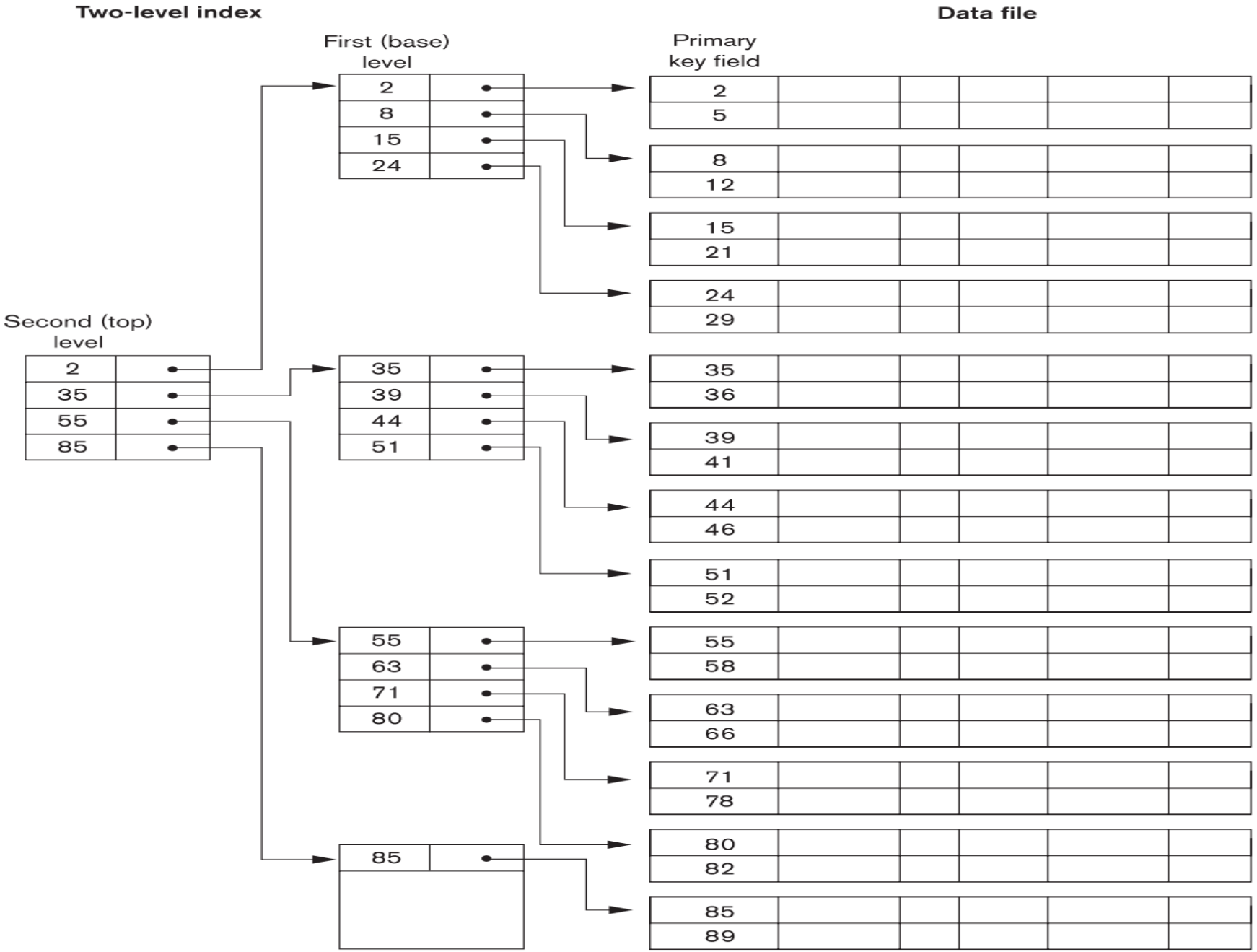
- The third level, which is a primary index for the second level, has an entry for each second-level block. Repeat the preceding process until all the entries of some index level fit in a single block.

- A multilevel index reduces the number of blocks accessed when searching for a record, given its indexing field value.

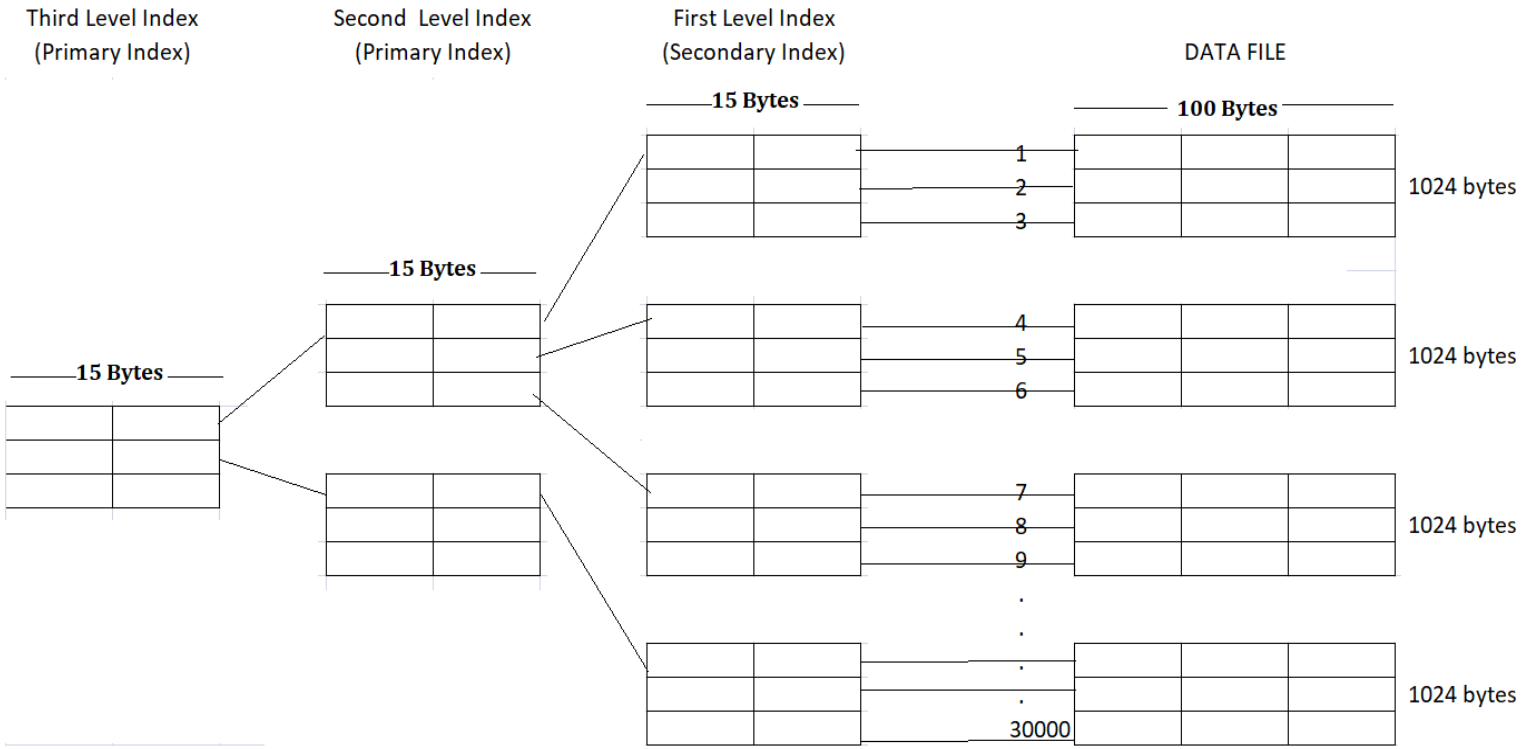
- All index levels are physically ordered files.
- To retain the benefits of using multilevel indexing while reducing index insertion and deletion problems, designers adopted a multilevel index called a dynamic multilevel index that leaves some space in each of its blocks for inserting new entries.


DATA FILE
Third Level Index
(Primary Index)
Second Level Index
(Primary Index)
First Level Index
(Primary Index)
15 Bytes
15 Bytes
15 Bytes
100 Bytes
1024 bytes
1024 bytes
1024 bytes
1024 bytes
1024 bytes
1024 bytes
1024 bytes

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



- Suppose an ordered file **with 30,000** records stored on a disk with block size **1024 bytes**. File records are of fixed size with record length **R = 100 bytes**. Indexing Fileds have ordering key field of the file is **V = 9 bytes** long, a blockpointer is **P = 6 bytes** long. The first level inex is supposed to be secondary Index and 2nd and 3rd level indexes are of Primary Index.
- Find
  - the blocking factor
  - no.of blocks in each index levels
  - no. of block access



- Blocking factor in first level index =

No of record per block=

$\lfloor$ blocksize /record size $\rfloor$

= $\lfloor$ 1024/15 $\rfloor$ = $\lfloor$ 68.2 $\rfloor$ = 68 records

No. of blocks in First level index =

$\lceil$ No. of records in index file / blocking factor $\rceil$

Since first level index is secondary index, No. of records in index file = no. of records in data file

= $\lceil$ 30000/68 $\rceil$ = $\lceil$ 441.17 $\rceil$ = 442

No. of blocks in Second level index =

⌈ No. of records in index file / blocking factor ⌉

Since second level index is primary index,

No. of records in primary index file = no. of blocks in first level index

= ⌈ 442/68 ⌉ = ⌈ 6.5 ⌉ = 7 blocks

No. of blocks in Third level index =

⌈ No. of records in index file / blocking factor ⌉

Since third level index is primary index, No. of records in third level index file = no. of blocks in first level index

= ⌈ 7/68 ⌉ = ⌈ 0.10 ⌉ = 1 block

- No of block access = no. of levels + 1

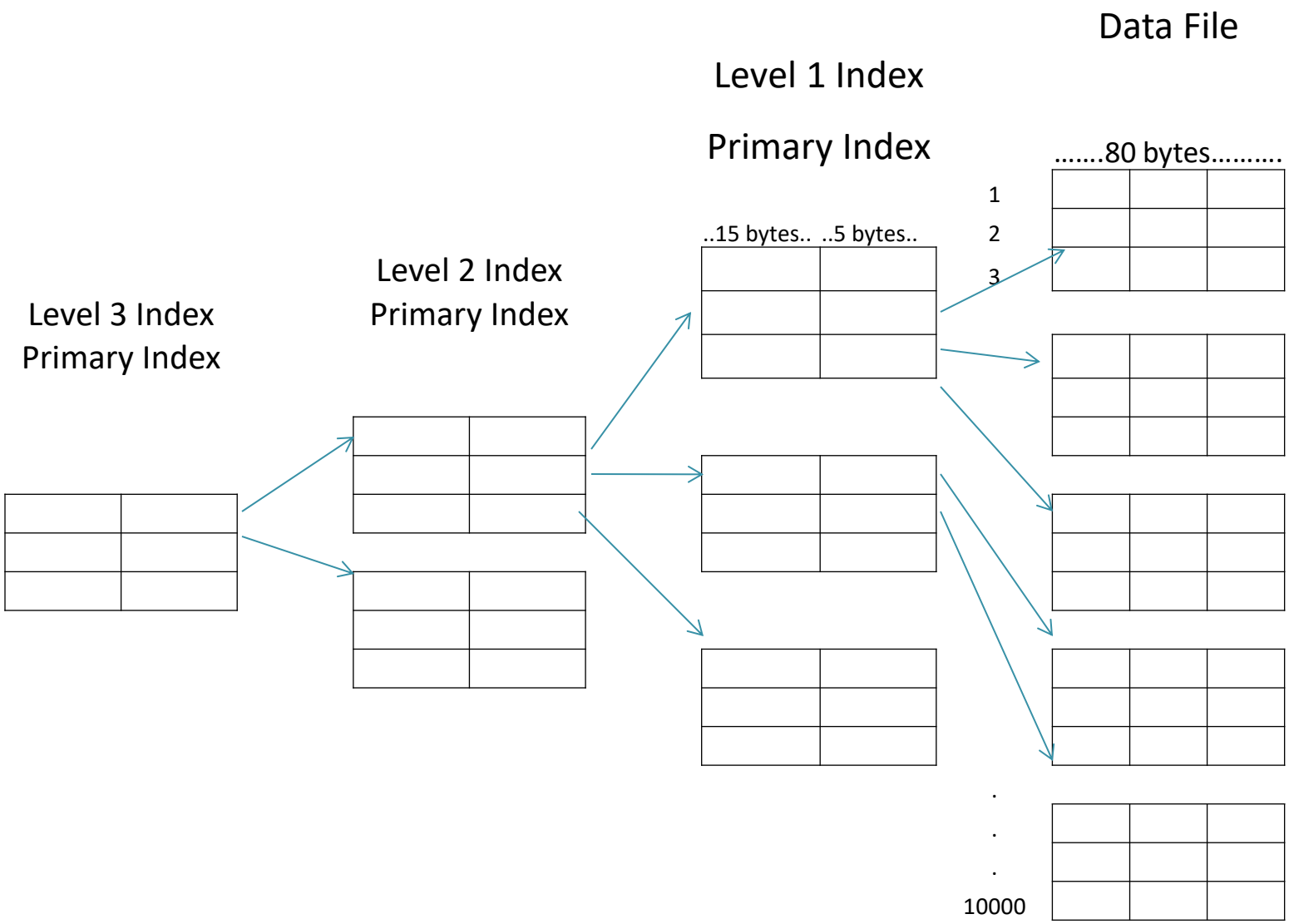
  = 3 + 1 = 4 block access

Database Management System
Module 3

Lect 13 :
University questions on INDEX STRUCTURES

University problems on Indexing

Consider an EMPLOYEE file with 10000 records where each record is of size 80 bytes. The file is sorted on employee number (15 bytes long), which is the primary key. Assuming un-spanned organization, block size of 512 bytes and block pointer size of 5 bytes, compute the number of block accesses needed for retrieving an employee record based on employee number if (i) No index is used (ii)Multi-level primary index is used


Data File
Level 1 Index
Primary Index
.....80 bytes.....
1
2
3
..15 bytes..
..5 bytes..
Level 2 Index
Primary Index
Level 3 Index
Primary Index
.
.
.
10000

(i)No Index used

Block size = 512 bytes

No. of records= 1000

Record size = 80 bytes

Blocking Factor = $\lfloor$ blocksize /record size $\rfloor$

= $\lfloor$ **512/80** $\rfloor$ = $\lfloor$ **6.4** $\rfloor$ **= 6 records**

No. of blocks in data file =

$\lceil$ No. of records in index file / blocking factor $\rceil$

= $\lceil$ 10000/ 6$\rceil$ = $\lceil$ 1666.66 $\rceil$ = 1667 blocks

---

No of block access **= $\lceil \log_2 B \rceil$ =**

$\lceil \log_2 1667 \rceil$ = 11 block access

(ii)Multi-level primary index is used

Block size = 512 bytes

Key field = 15 bytes

Block pointer = 5 bytes

Record size = 15 + 5=20 bytes

Blocking Factor =

$\lfloor$ blocksize /record size $\rfloor$

= $\lfloor$ **512/20** $\rfloor$ = $\lfloor$ **25.6** $\rfloor$ **= 25 records**

No. of blocks in First level Index =

⌈ No. of records in First level index file / blocking factor ⌉

Since First level Index is a Primary Index,

No. of records in First level index file = No. of blocks in data file = 1667

No. of blocks in First level Index =

⌈ 1667/ 25 ⌉ = ⌈ 66.68⌉ = 67 blocks

No. of blocks in Second level Index =

⌈ No. of records in Second level index file / blocking factor ⌉

Since Second level Index is a Primary Index,
No. of records in Second level index file =
No. of blocks in First level index= 67

No. of blocks in Second level Index =

⌈ 67/ 25 ⌉ = ⌈2.68 ⌉ = 3 blocks

No. of blocks in Third level Index =

⌈ No. of records in Third level index file / blocking factor ⌉

Since Third level Index is a Primary Index,
No. of records in Third level index file = No. of blocks in Second level index= 3

No. of blocks in Second level Index =

⌈ 3/ 25 ⌉ = ⌈0. 12⌉ = 1 block

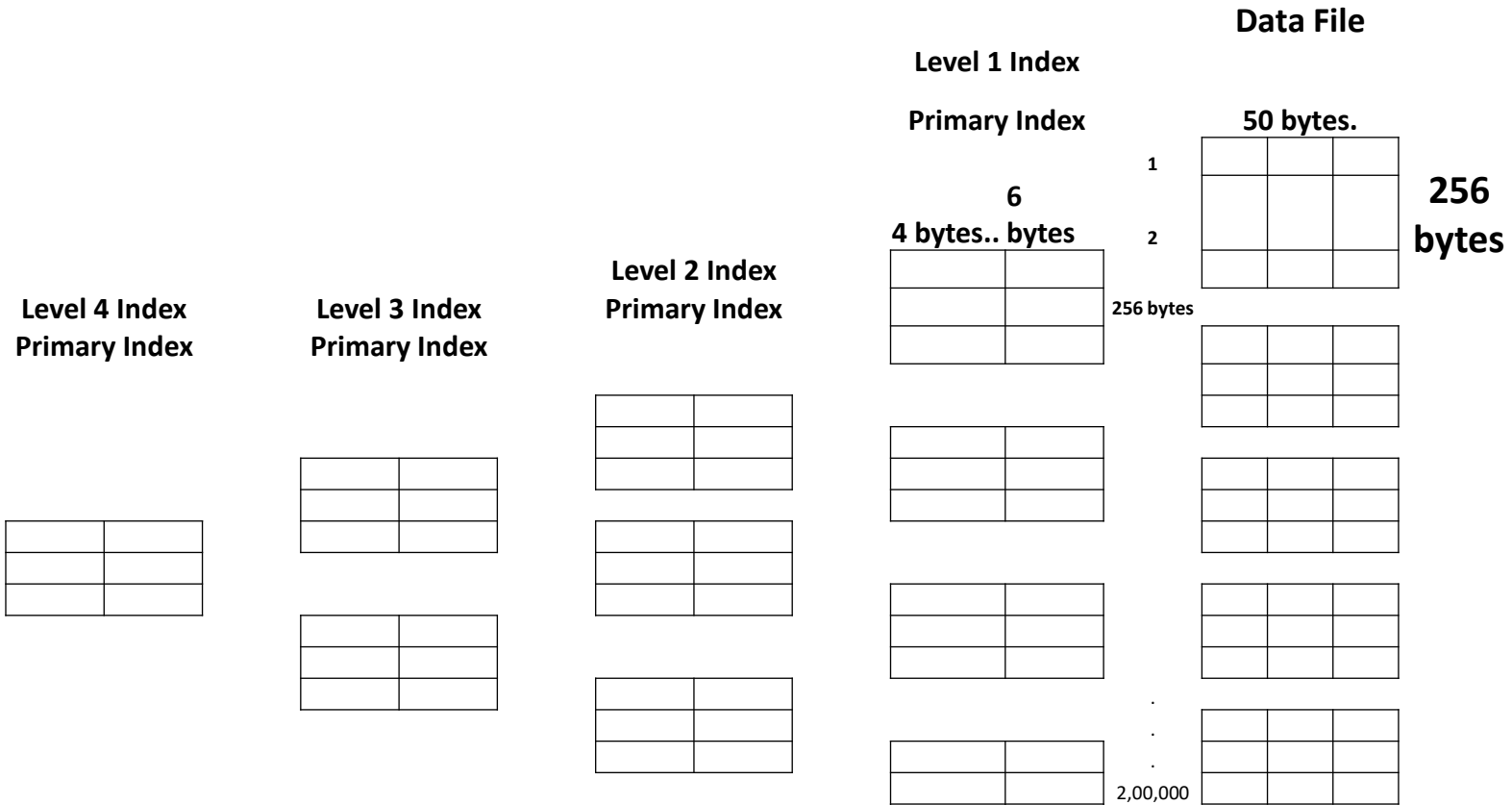
- No. of block access required to access the record = no. of levels(t) + 1

No. of levels = 3

No. of block access required to access the record = 3 + 1 = 4 block access.

Consider a file with 2,00,000 records stored in a disk with fixed length blocks of size 256 bytes. Each record is of size 50 bytes. The primary key is 4 bytes and block pointer is 6 bytes. Compute the following, assuming that multi-level primary index is used as access path: (10)

(i) Blocking factor for data records
(ii) Blocking factor for index records
(iii) Number of data blocks
(iv) Number of First level index blocks
(v) Number of levels of multi level index



(i) Blocking Factor of data record

Block size = 256 bytes

Record size = 50 bytes

Blocking Factor =

$\lfloor$ blocksize /record size $\rfloor$

= $\lfloor$ **256/50** $\rfloor$ = $\lfloor$ **5.12** $\rfloor$ **= 5 records**

(ii) Blocking Factor of Index record

Block size = 256 bytes

Record size = 4+6 = 10 bytes

Blocking Factor =

$\lfloor$ blocksize /record size $\rfloor$

= $\lfloor$ **256/10** $\rfloor$ **=** $\lfloor$ **25.6** $\rfloor$ **= 25 records**

(iii) Number of data blocks

=⌈ No. of records in data file / blocking factor of data file ⌉

= ⌈ 200000/ 5⌉ = ⌈ 40000 ⌉ = 40000blocks

(iv) Number of first level index blocks

$\lceil$ No. of records in index file / blocking factor of index file $\rceil$

No. of records in first level index file= no. of blocks in data file

$\lceil$ 40000/ 25 $\rceil$ = 1600 blocks

(v) No. of levels of multilevel Index

- Number of second level index blocks
  - ⌈ No. of records in first level index file / blocking factor of index file⌉
  - ⌈ 1600/ 25 ⌉ = 64 blocks

- Number of third level index blocks
  - ⌈ No. of records in third level index file / blocking factor of index file⌉
  - ⌈ 64/ 25 ⌉ = 3 blocks

- Number of fourth level index blocks
  - ⌈ No. of records in fourth level index file / blocking factor of index file⌉
  - ⌈ 3/ 25 ⌉ = ⌈ 0.12 ⌉ = 1 block

- No. of levels of multilevel index = 4

There are 12000 records in a data file. Each record in the file is of 75 bytes. (7)
Compute the number of block accesses if (i) Single level secondary index is available on a field of size 15 bytes. (ii) Multilevel index is available on the same field.
Assume that the block size is 394 bytes, that un-spanned organization is used and that block and record pointers are 5 and 7 bytes, respectively.

(i) If Single level secondary Index is used

Block size= 394 bytes

Record size of index file = 5 + 7 =12 bytes

Blocking Factor = $\lfloor$ blocksize /record size $\rfloor$

= $\lfloor$ **394/12** $\rfloor$ = $\lfloor$ **32.8** $\rfloor$ **= 32 records**

Number of index blocks

=$\lceil$No. of records in index file / blocking factor of index file $\rceil$

= $\lceil$ **12000/ 32** $\rceil$ **= 375 blocks**

Number of block access = $\lceil \log_2 B \rceil = \lceil \log_2 375 \rceil = \lceil 8.55 \rceil$

= 9+1=10

(i) If Multi level Index is used

Block size= 394 bytes

Record size of index file = 5 + 7 =12 bytes

Number of first level index blocks

=⌈No. of records in first level index file / blocking factor of index file ⌉

= ⌈**12000/ 32**⌉ **= 375 blocks**

Number of second level index blocks

= ⌈375/ 32⌉ = ⌈11.71⌉ = 12 blocks

Number of third level index blocks

= ⌈12/ 32⌉ = ⌈0.375⌉ = 1 block

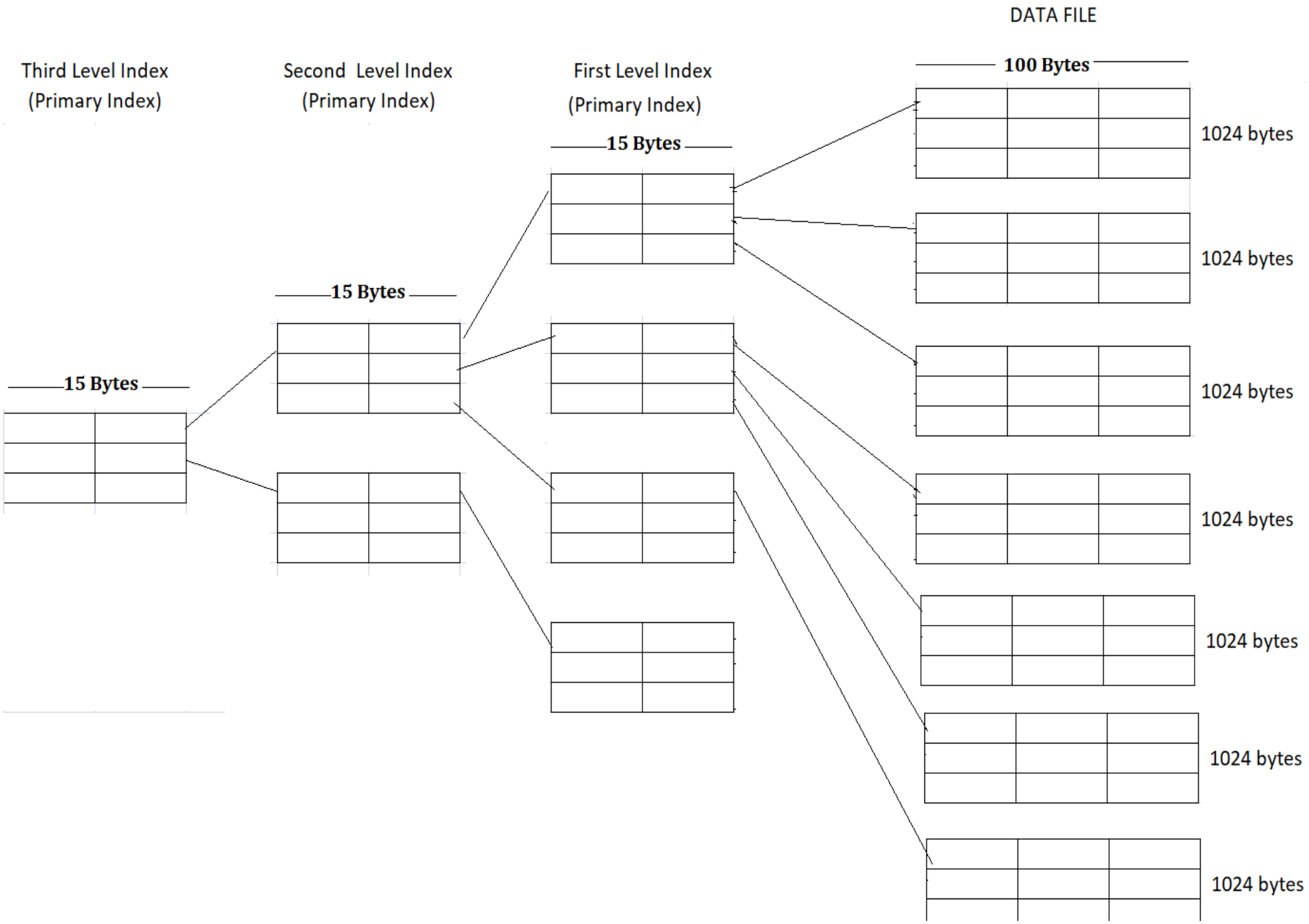
Number of block access = No of levels (t) + 1 = **3+ 1 = 4 block access**

TYPES OF INDEXING

- Single level ordered indexes
  - Primary index
  - Clustering index
  - Secondary index
- Multilevel index
- Dynamic Multilevel Indexes Using B+-Trees

Database Management System
Module 3

Lect 14 :
DYNAMIC MULTI LEVEL INDEX-
B+ Tree


DATA FILE
Third Level Index
(Primary Index)
Second Level Index
(Primary Index)
First Level Index
(Primary Index)
15 Bytes
15 Bytes
15 Bytes
100 Bytes
1024 bytes
1024 bytes
1024 bytes
1024 bytes
1024 bytes
1024 bytes
1024 bytes

Dynamic Multilevel Indexes Using B+Trees

- B+ trees are special cases of search tree.
- A tree is formed of nodes.
- Each node except the root, has one parent node and zero or more child nodes.
- The root node has no parent.
- A node that does not have any child nodes is called a leaf node; a non-leaf node is called an internal node.

- The level of a leaf node is always one more than the level of its parent,
- level of the root node is zero.
- A sub-tree of a node consists of that node and all its descendant nodes
- B+ tree ensures that the tree is always balanced and that the space wasted by deletion never becomes excessive.
- Insertions and deletions are simple processes.
- Structure of the internal node and leaf node of a B+ tree is different.

Structure of the internal nodes of a B+ tree of Order 'p'

1. Each internal node is of the form

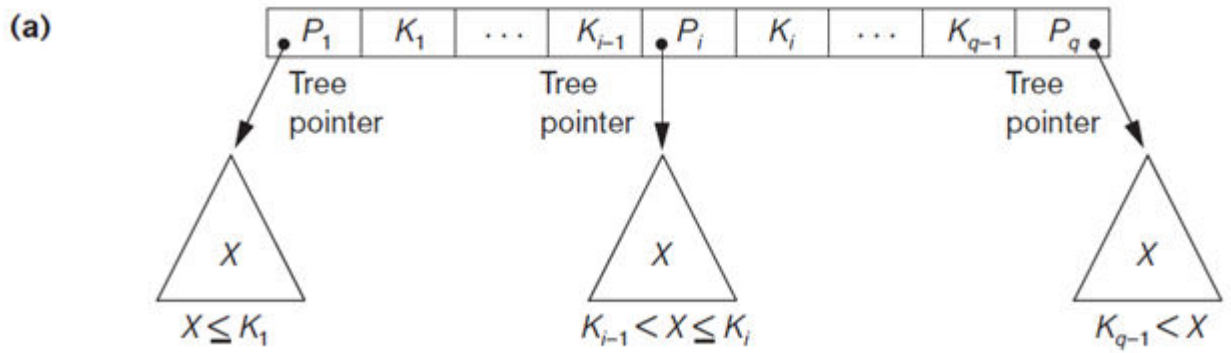
$$<P_1, K_1, P_2, K_2, ..., P_{q-1}, K_{q-1}, P_q>$$

where $q \leq p$ and each P_i is a **tree pointer**.

2. Within each internal node,

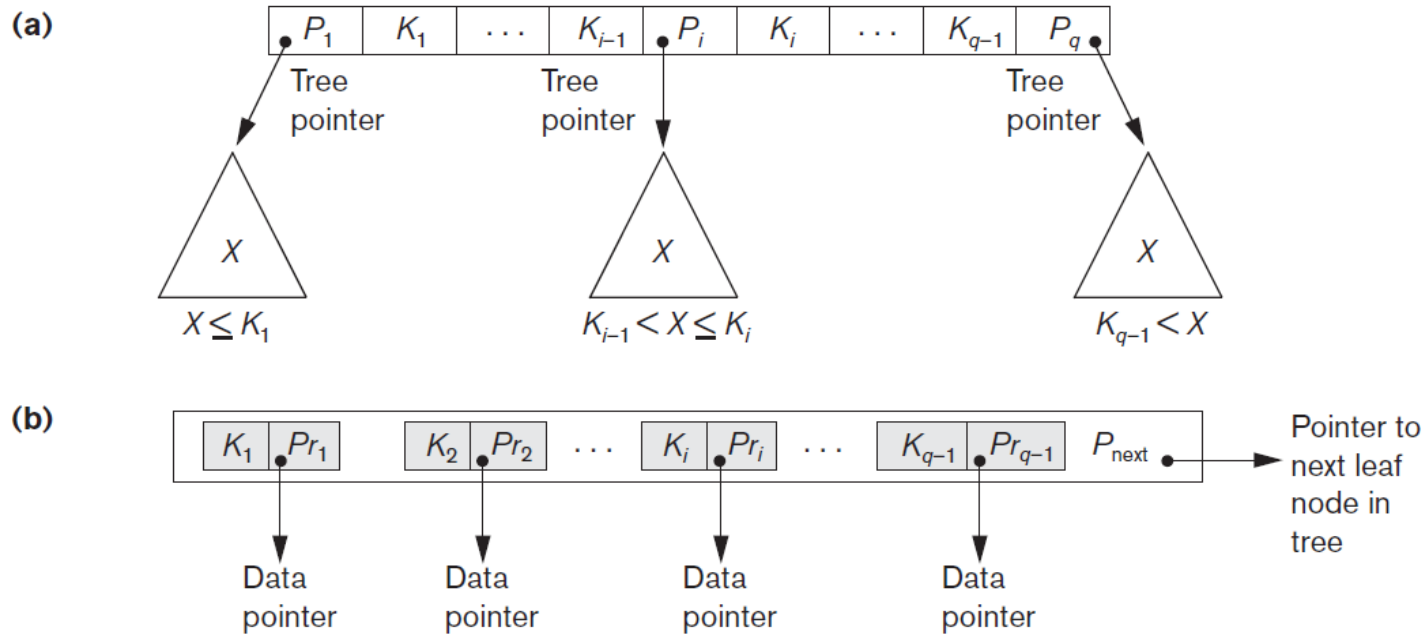
$$\mathbf{K_1 < K_2 < ... < K_{q-1}}.$$

3. Each internal node has at most **p** tree pointers
4. Each internal node, except the root, has at least $\lceil (p/2) \rceil$ tree pointers. The root node has at least two tree pointers if it is an internal node



5. For all search field values X in the subtree pointed at by P_i, we have:
   - $K_{i-1} < X \leq K_i$
   - $X \leq K_i$
   - $X > K_{i-1}$

The nodes of a B+-tree. (a) Internal node of a B+-tree with $q - 1$ search values. (b) Leaf node of a B+-tree with $q - 1$ search values and $q - 1$ data pointers.



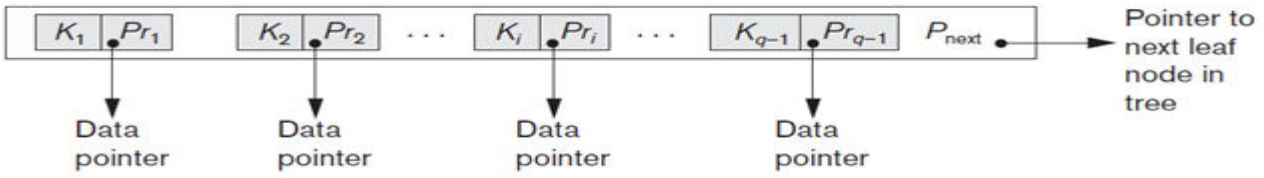
Structure of the leaf nodes of a B+ tree of order p is

- Each leaf node is of the form

$$<<K_1, Pr_1>, <K_2, Pr_2>, ..., <K_{q-1}, Pr_{q-1}>, P_{next}>$$

where $q \leq p$, each Pr_i is a data pointer, and P_{next} points to the next leaf node of the B+ tree.

- Within each leaf node, $K_1 \leq K_2 ... , K_{q-1}$, $q \leq p$.
- Each Pr_i is a data pointer that points to the record whose search field value is K_i or to a file block containing the record
- Each leaf node has at least $\lceil (p/2) \rceil$ values.
- All leaf nodes are at the same level

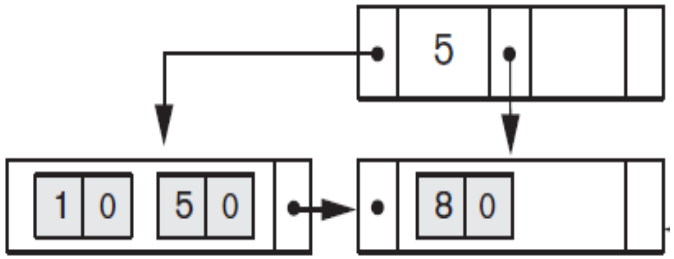


B+ Tree Insertion

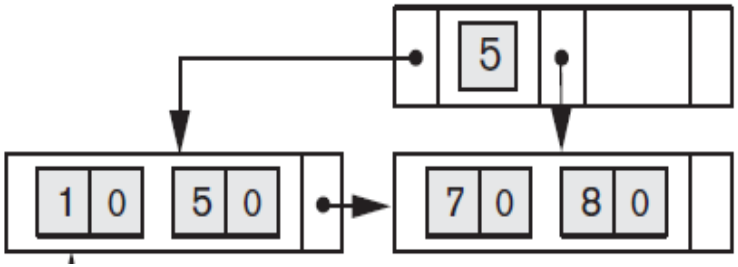
- Insert value into B+ Tree with order P=3
- No. of search key value= P-1 = 3-1=2
- Max no of tree pointers = P =3
- Min no of tree pointers = $\lceil (p/2) \rceil$ = 2

- Insert 5,8,1,7,3,12,9,6,
- Initially the tree is null.
- **Insert 5 and 8**

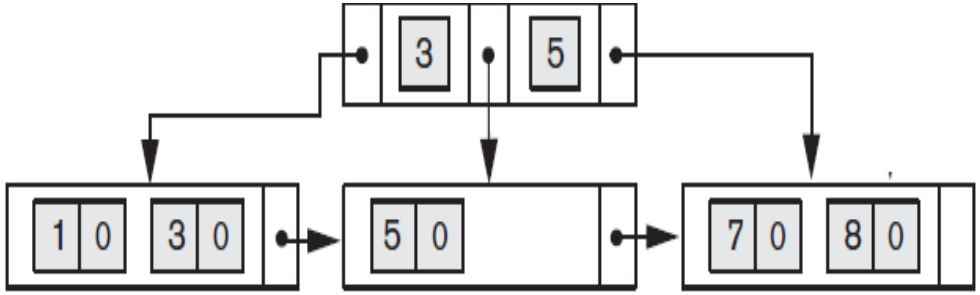
- **Insert 1:** The root node is full since it is leaf node now. So create a new level.



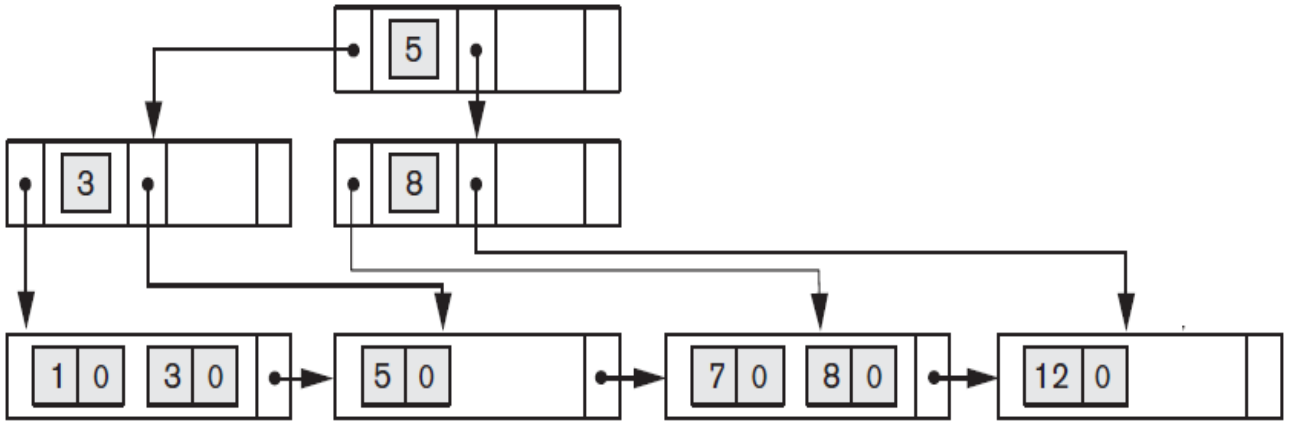
- **Insert 7:** It can be inserted in the 2nd leaf node.



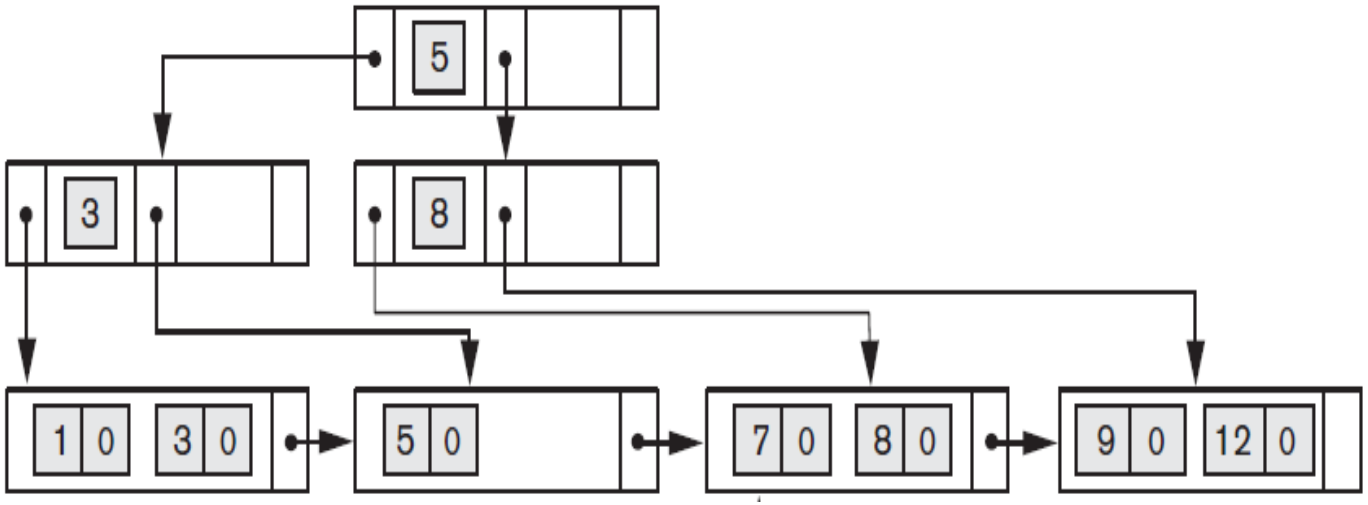
- **Insert 3:** It should be inserted in the 1st leaf node. But the 1St node is already full. So create new level



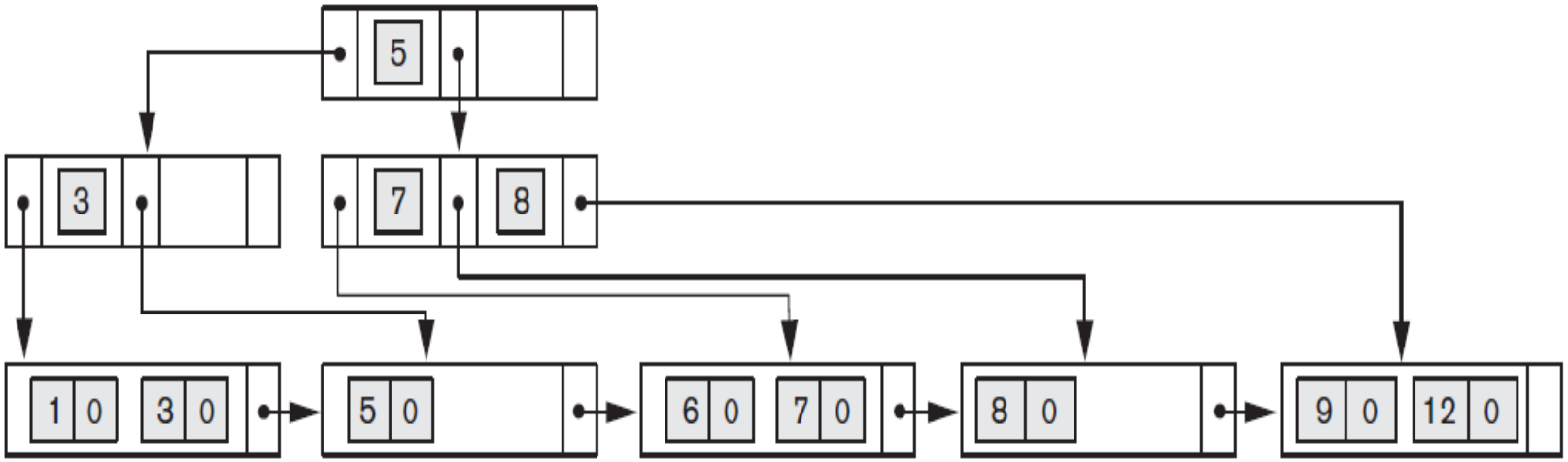
- **Insert 12:** It should be inserted in the last leaf node. But the node is already full. So create new node and 8 goes to the next higher level (2nd level). Again the node is full. So split the node. Then 5 goes to root node. 3 will be in one node and 8 will be in another node.



- **Insert 9:** It can be inserted in the last leaf node



- **Insert 6:** It should be inserted in the 3rd leaf node. But the node is already full. So create new node and 7 goes to the next higher level (2nd level).



Database Management System
Module 3

Lect 15 :
B+ Tree Deletion

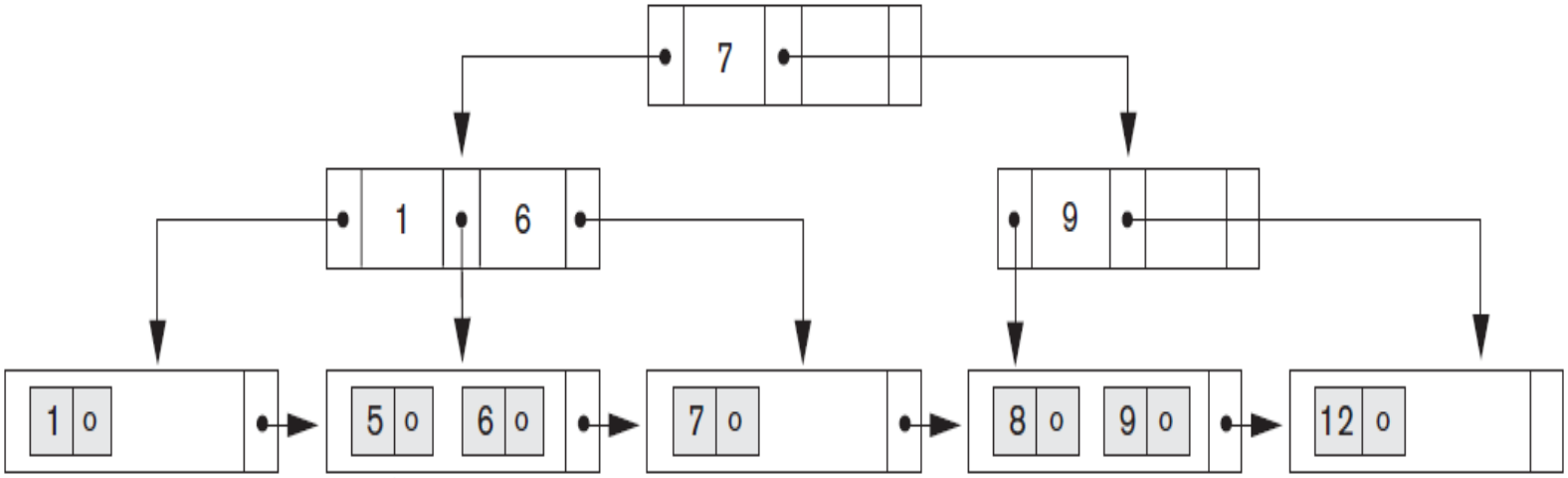
B + Tree Deletion

- When an entry is deleted, it is removed from the leaf level.
- If it happens to occur in an internal node, it must also be removed from there. And the leaf node must replace it in the internal node.
- Deletion may cause underflow by reducing the number of entries in the leaf node to below the minimum required. In this case, try to find a sibling leaf node—a leaf node directly to the left or to the right of the node with underflow—and redistribute the entries among the node and its sibling so that both are at least half full; otherwise, the node is merged with its siblings and the number of leaf nodes is reduced.
- A common method is to try to redistribute entries with the left sibling; if this is not possible, an attempt to redistribute with the right sibling is made. If this is also not possible, the three nodes are merged into two leaf nodes. In such a case, underflow may propagate to internal nodes because one fewer tree pointer and search value are needed. This can propagate and reduce the tree levels.

B+ Tree Deletion

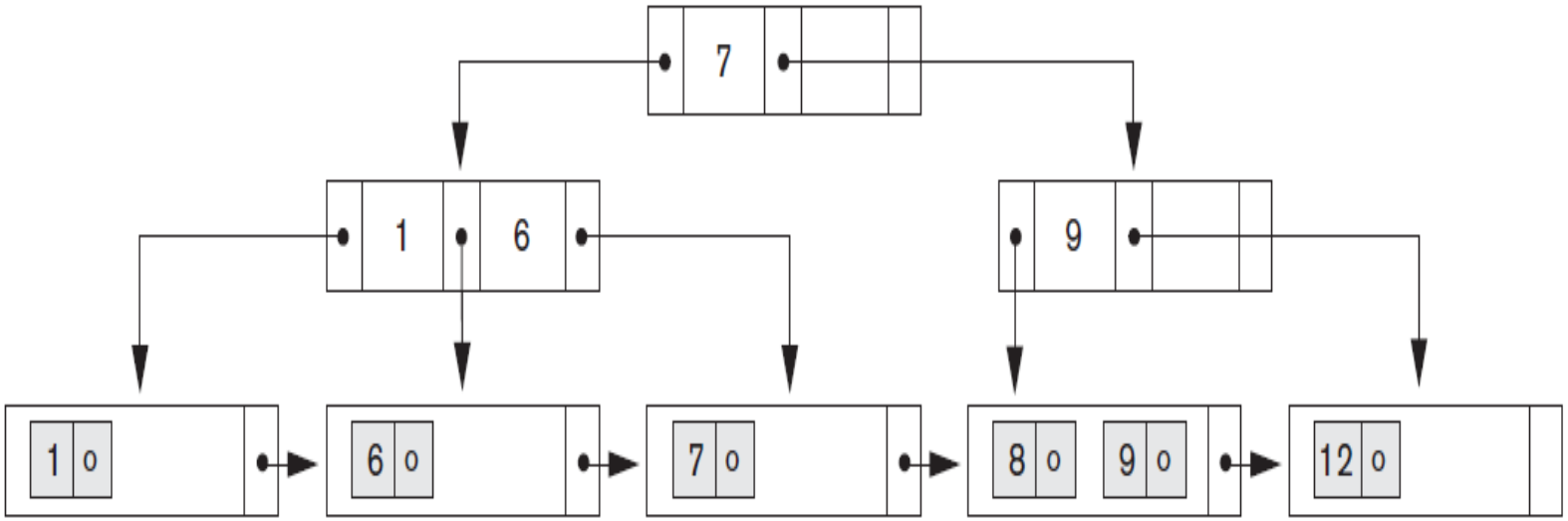
Example

- Order (p) =3
- No. of keys = p-1 = 3-1=2
- Min no of keys $\lceil p/2 \rceil$-1 =1

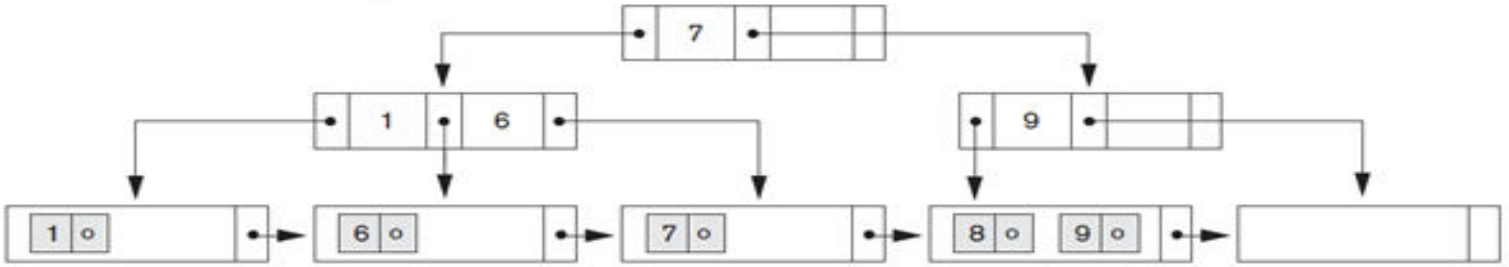


Delete 5

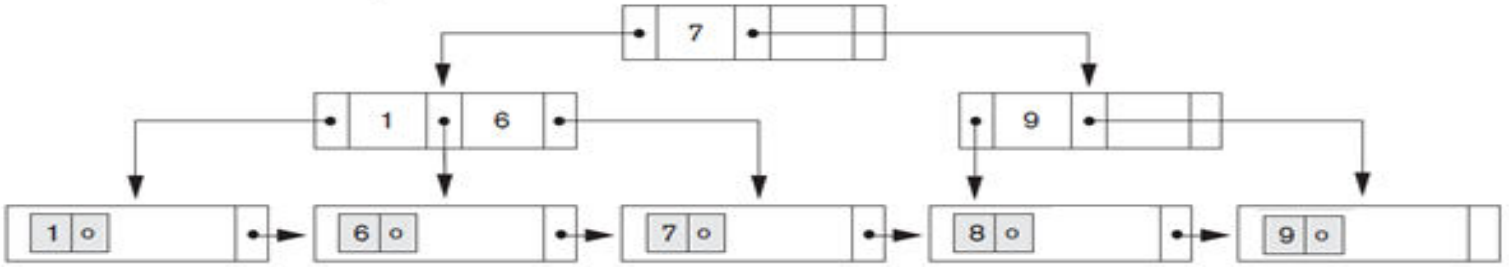
- It is only in the leaf node. So simply delete it



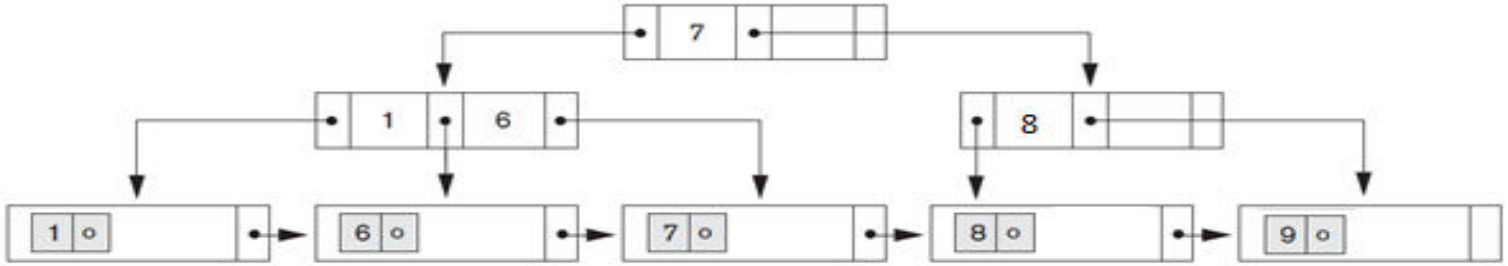
Delete 12



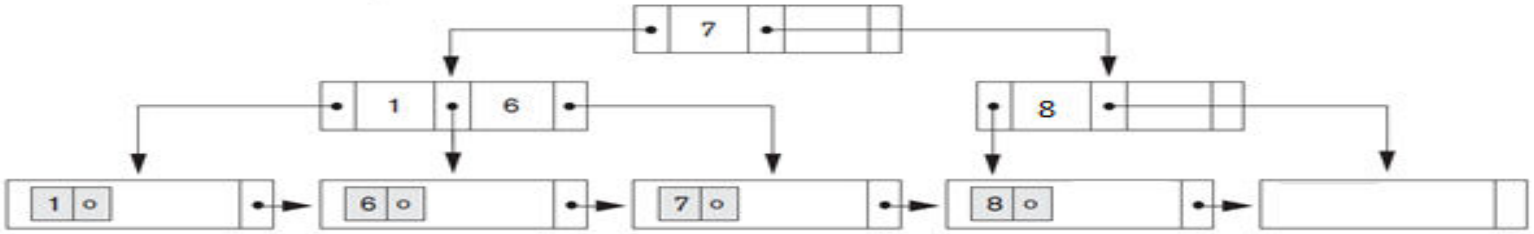
Leaf Node
Underflow



Not obeying
B+ tree rules



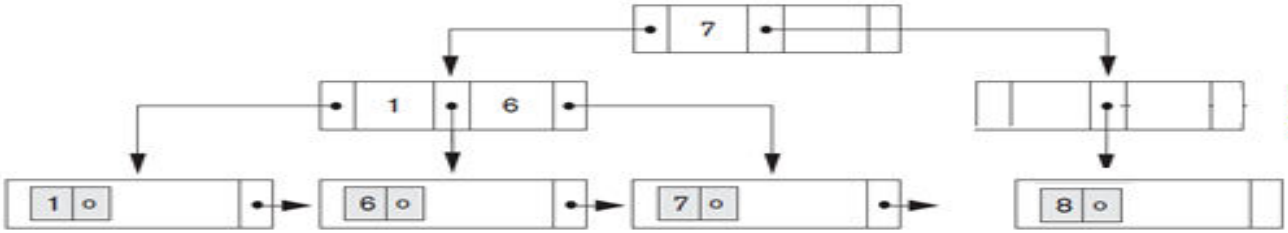
Delete 9



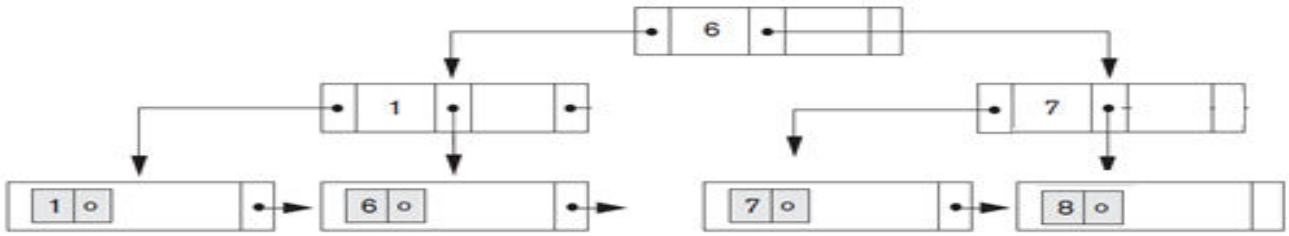
Leaf Node Underflow

Sibling have only min no. of search key

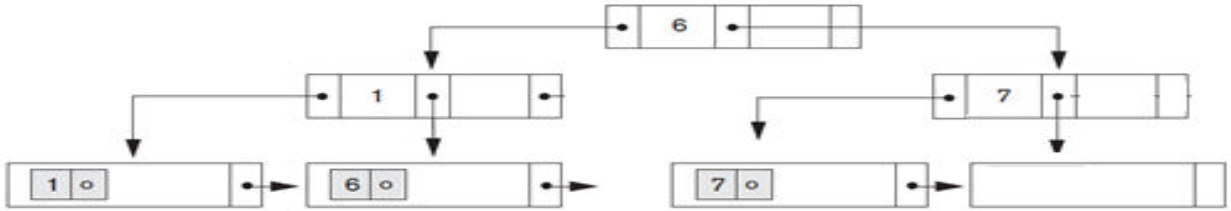
So Merge two Leaf nodes



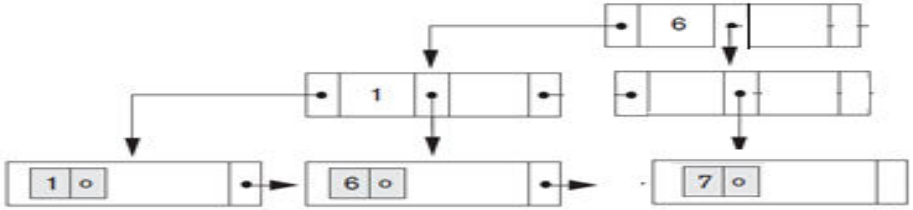
Node Underflow
Share from sibling



Delete 8



Leaf Node Underflow
Sibling have only min no of search keys
Merge leaf node



Node Underflow
Sibling have only min no of search keys
Merge node