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.

Basic Retrieval Queries in SQL

- The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- SQL has one basic statement for retrieving information from a database: the SELECT statement.
- SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values.
- Hence, in general, an SQL table is not a set of tuples, because a set does not allow two identical members; rather, it is a multiset (sometimes called a bag) of tuples.

The SELECT-FROM-WHERE Structure of Basic SQL Queries

The basic form of the SELECT statement, sometimes called a **mapping** or a **select-from-where block**, is formed of the three clauses SELECT, FROM, and WHERE and has the following form:

SELECT <attribute list>

FROM <table list>

WHERE <condition>;

where

- <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.

In SQL, the basic logical comparison operators for comparing attribute values with one another and with literal constants are =, <, <=, >, >=, and <>. These correspond to the relational algebra operators =, <, ≤, >, ≥, and ≠, respectively.

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

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

Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:   SELECT   Bdate, Address
      FROM     EMPLOYEE
      WHERE    Fname='John' AND Minit='B' AND Lname='Smith';
```

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

Bdate	Address
1965-01-09	731 Fondren, Houston, TX

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:   SELECT   Fname, Lname, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    Dname='Research' AND Dnumber=Dno;
```

- A query that involves only selection and join conditions plus projection attributes is known as a select-project-join query. The next example is a select-project-join query with two join conditions.

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:   SELECT   Pnumber, Dnum, Lname, Address, Bdate
      FROM     PROJECT, DEPARTMENT, EMPLOYEE
      WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND
               Plocation='Stafford';
```

Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- In SQL, the same name can be used for two (or more) attributes as long as the attributes are in different relations.
- If this is the case, and a multitable query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name to prevent ambiguity.
- This is done by prefixing the relation name to the attribute name and separating the two by a period

Suppose that the Dno and Lname attributes of the EMPLOYEE relation were called Dnumber and Name, and the Dname attribute of DEPARTMENT was also called Name; then, to prevent ambiguity, Query 1 would be rephrased as shown in Q1A.

```
Q1A:  SELECT   Fname, EMPLOYEE.Name, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DEPARTMENT.Name='Research' AND
               DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

We can also create an *alias* for each table name to avoid repeated typing of long table names. E and S, called aliases or tuple variables, for the EMPLOYEE relation. An alias can follow the keyword 'AS'

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
Q8:   SELECT   E.Fname, E.Lname, S.Fname, S.Lname
      FROM     EMPLOYEE AS E, EMPLOYEE AS S
      WHERE    E.Super_ssn=S.Ssn;
```

Unspecified WHERE Clause and Use of the Asterisk

- A missing WHERE clause indicates no condition on tuple selection; hence, all tuples of the relation specified in the FROM clause qualify and are selected for the query result.
- If more than one relation is specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT— all possible tuple combinations—of these relations is selected

Queries 9 and 10. Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

Q9:
```
SELECT  Ssn
FROM    EMPLOYEE;
```

Ssn
123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555

Q10:
```
SELECT  Ssn, Dname
FROM    EMPLOYEE, DEPARTMENT;
```

(f)

Ssn	Dname
123456789	Research
333445555	Research
999887777	Research
987654321	Research
666884444	Research
453453453	Research
987987987	Research
888665555	Research
123456789	Administration
333445555	Administration
999887777	Administration
987654321	Administration
666884444	Administration
453453453	Administration
987987987	Administration
888665555	Administration
123456789	Headquarters
333445555	Headquarters
999887777	Headquarters
987654321	Headquarters
666884444	Headquarters
453453453	Headquarters
987987987	Headquarters
888665555	Headquarters

- To retrieve all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL;
- We just specify an asterisk (*), which stands for all the attributes

For example, query Q1C retrieves all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5,
query Q1D retrieves all the attributes of an EMPLOYEE and the attributes of the DEPARTMENT in which he or she works for every employee of the 'Research' department,
and Q10A specifies the CROSS PRODUCT of the EMPLOYEE and DEPARTMENT relations.

```
Q1C:    SELECT    *
        FROM      EMPLOYEE
        WHERE     Dno=5;

Q1D:    SELECT    *
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     Dname='Research' AND Dno=Dnumber;

Q10A:   SELECT    *
        FROM      EMPLOYEE, DEPARTMENT;
```

Tables as Sets in SQL

- SQL usually treats a table not as a set but rather as a multiset; duplicate tuples can appear more than once in a table, and in the result of a query.
- SQL does not automatically eliminate duplicate tuples in the results of queries, for the following reasons:
  - Duplicate elimination is an expensive operation. One way to implement it is to sort the tuples first and then eliminate duplicates.
  - The user may want to see duplicate tuples in the result of a query.
  - When an aggregate function is applied to tuples, in most cases we do not want to eliminate duplicates.

- If we do want to eliminate duplicate tuples from the result of an SQL query, we use the keyword DISTINCT in the SELECT clause, meaning that only distinct tuples should remain in the result.
- In general, a query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not.
- Specifying SELECT with neither ALL nor DISTINCT—as in our previous examples— is equivalent to SELECT ALL

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

```
Q11:    SELECT    ALL Salary
        FROM      EMPLOYEE;

Q11A:   SELECT    DISTINCT Salary
        FROM      EMPLOYEE;
```

(a)

Salary
30000
40000
25000
43000
38000
25000
25000
55000

(b)

Salary
30000
40000
25000
43000
38000
55000

Figure 4.4
Results of additional SQL queries when applied to the COMPANY database state shown in Figure 3.6. (a) Q11. (b) Q11A.

- SQL has directly incorporated some of the set operations from mathematical set theory, which are also part of relational algebra.
- There are set union (UNION), set difference (EXCEPT), and set intersection (INTERSECT) operations.
- The relations resulting from these set operations are sets of tuples; that is, duplicate tuples are eliminated from the result.
- These set operations apply only to union-compatible relations, so we must make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations.
- SQL also has corresponding multiset operations, which are followed by the keyword ALL (UNION ALL, EXCEPT ALL, INTERSECT ALL). Their results are multisets (duplicates are not eliminated).

Q4. Make a list of all project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project

```
Q4A:  (SELECT   DISTINCT Pnumber
       FROM     PROJECT, DEPARTMENT, EMPLOYEE
       WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn
                AND Lname='Smith' )
       UNION
      ( SELECT  DISTINCT Pnumber
       FROM     PROJECT, WORKS_ON, EMPLOYEE
       WHERE    Pnumber=Pno AND Essn=Ssn
                AND Lname='Smith' );
```

The first SELECT query retrieves the projects that involve a ‘Smith’ as manager of the department that controls the project, and the second retrieves the projects that involve a ‘Smith’ as a worker on the project. Notice that if several employees have the last name ‘Smith’, the project names involving any of them will be retrieved. Applying the UNION operation to the two SELECT queries gives the desired result.

Substring Pattern Matching and Arithmetic Operators

- The first feature allows comparison conditions on only parts of a character string, using the **LIKE** comparison operator.
- This can be used for string pattern matching.
- Partial strings are specified using two reserved characters: **%** replaces an arbitrary number of zero or more characters, and the underscore (_) replaces a single character.

Query 12. Retrieve all employees whose address is in Houston, Texas.

```
Q12:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Address LIKE '%Houston,TX%';
```

To retrieve all employees who were born during the 1950s, we can use Query Q12A. Here, '5' must be the third character of the string (according to our format for date), so we use the value '_ _ 5 _ _ _ _ _ _ _', with each underscore serving as a placeholder for an arbitrary character.

Query 12A. Find all employees who were born during the 1950s.

```
Q12:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Bdate LIKE '_ _ 5 _ _ _ _ _ _ _';
```

- If an underscore or % is needed as a literal character in the string, the character should be preceded by an escape character, which is specified after the string using the keyword ESCAPE.
- For example, 'AB_CD\%EF' ESCAPE '\' represents the literal
- string 'AB_CD%EF' because \ is specified as the escape character.
- Any character not used in the string can be chosen as the escape character.
- Also, we need a rule to specify apostrophes or single quotation marks (' ') if they are to be included in a string because they are used to begin and end strings.
- If an apostrophe (') is needed, it is represented as two consecutive apostrophes (") so that it will not be interpreted as ending the string.

- The standard arithmetic operators for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains

Query 13. Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
Q13:  SELECT   E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
      FROM     EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
      WHERE    E.Ssn=W.Essn AND W.Pno=P.Pnumber AND
               P.Pname='ProductX';
```

Query 14. Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

```
Q14:  SELECT   *
      FROM     EMPLOYEE
      WHERE    (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

The condition (Salary **BETWEEN** 30000 **AND** 40000) in Q14 is equivalent to the condition ((Salary >= 30000) **AND** (Salary <= 40000)).

Ordering of Query Results

- The ORDER BY statement in sql is used to sort the fetched data in either ascending or descending according to one or more columns.
- By default ORDER BY sorts the data in ascending order.
- We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

SELECT * FROM Student ORDER BY ROLL_NO ASC;

Student Table

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18

Query 15. Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.

```
Q15:    SELECT      D.Dname, E.Lname, E.Fname, P.Pname
        FROM        DEPARTMENT D, EMPLOYEE E, WORKS_ON W,
                    PROJECT P
        WHERE       D.Dnumber= E.Dno AND E.Ssn= W.Essn AND
                    W.Pno= P.Pnumber
        ORDER BY    D.Dname, E.Lname, E.Fname;
```

Nested Queries, Tuples, and Set/Multiset Comparisons

- Some queries require that existing values in the database be fetched and then used in a comparison condition.
- Such queries can be conveniently formulated by using nested queries, which are complete select-from-where blocks within the WHERE clause of another query. That other query is called the outer query

```
Q4A:  SELECT   DISTINCT Pnumber
      FROM     PROJECT
      WHERE    Pnumber IN
               ( SELECT     Pnumber
                 FROM       PROJECT, DEPARTMENT, EMPLOYEE
                 WHERE      Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
               OR
               Pnumber IN
               ( SELECT     Pno
                 FROM       WORKS_ON, EMPLOYEE
                 WHERE      Essn=Ssn AND Lname='Smith' );
```

- The first nested query selects the project numbers of projects that have an employee with last name ‘Smith’ involved as manager, while the second nested query selects the project numbers of projects that have an employee with last name ‘Smith’ involved as worker.

- In the outer query, we use the **OR** logical connective to retrieve a PROJECT tuple if the PNUMBER value of that tuple is in the result of either nested query.

SQL allows the use of **tuples** of values in comparisons by placing them within parentheses. To illustrate this, consider the following query:

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```

This query will select the Essns of all employees who work the same (project, hours) combination on some project that employee 'John Smith' (whose Ssn = '123456789') works on. In this example, the IN operator compares the subtuple of values in parentheses (Pno, Hours) within each tuple in WORKS_ON with the set of type-compatible tuples produced by the nested query.

- The keyword ALL can be combined with each of >, >=, <, <=, and <>, these operators. For example, the comparison condition (v > ALL V) returns TRUE if the value v is greater than all the values in the set (or multiset) V.

An example is the following query, which returns the names of employees whose salary is greater than the salary of all the employees in department 5:

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL   ( SELECT    Salary
                           FROM      EMPLOYEE
                           WHERE     Dno=5 );
```

Correlated Nested Queries

- Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated.
- For each EMPLOYEE tuple, evaluate the nested query, which retrieves the Essn values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple; if the Ssn value of the EMPLOYEE tuple is in the result of the nested query, then select that EMPLOYEE tuple.

```
Q16A:  SELECT   E.Fname, E.Lname
       FROM     EMPLOYEE AS E, DEPENDENT AS D
       WHERE    E.Ssn=D.Essn AND E.Sex=D.Sex
                AND E.Fname=D.Dependent_name;
```

The EXISTS and UNIQUE Functions in SQL

- The EXISTS function in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.
- The result of EXISTS is a Boolean value TRUE if the nested query result contains at least one tuple, or FALSE if the nested query result contains no tuples
- EXISTS and NOT EXISTS are typically used in conjunction with a correlated nested query.

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16B:  SELECT   E.Fname, E.Lname
       FROM     EMPLOYEE AS E
       WHERE    EXISTS ( SELECT   *
                         FROM     DEPENDENT AS D
                         WHERE    E.Ssn=D.Essn AND E.Sex=D.Sex
                                  AND E.Fname=D.Dependent_name);
```

Query 6. Retrieve the names of employees who have no dependents.

```
Q6:    SELECT   Fname, Lname
       FROM     EMPLOYEE
       WHERE    NOT EXISTS ( SELECT   *
                             FROM     DEPENDENT
                             WHERE    Ssn=Essn );
```

Explicit Sets and Renaming of Attributes in SQL

- It is also possible to use an explicit set of values in the WHERE clause, rather than a nested query. Such a set is enclosed in parentheses in SQL

Query 17. Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3.

```
Q17:  SELECT  DISTINCT Essn
      FROM    WORKS_ON
      WHERE   Pno IN (1, 2, 3);
```

- In SQL, it is possible to rename any attribute that appears in the result of a query by adding the qualifier AS followed by the desired new name
- For example, Q8A shows the query to retrieve the last name of each employee and his or her supervisor, while renaming the resulting attribute names as Employee_name and Supervisor_name.
- The new names will appear as column headers in the query result.

```
Q8A:  SELECT  E.Lname AS Employee_name, S.Lname AS Supervisor_name
      FROM    EMPLOYEE AS E, EMPLOYEE AS S
      WHERE   E.Super_ssn=S.Ssn;
```

Joined Tables in SQL and Outer Joins

- The concept of a joined table (or joined relation) was incorporated into SQL to permit users to specify a table resulting from a join operation in the FROM clause of a query.
- For example, consider query Q1, which retrieves the name and address of every employee who works for the 'Research' department.
- It may be easier to specify the join of the EMPLOYEE and DEPARTMENT relations first, and then to select the desired tuples and attributes.
- This can be written in SQL as in Q1A:

```
Q1A:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
        WHERE     Dname='Research';
```

- The concept of a joined table also allows the user to specify different types of join, such as NATURAL JOIN and various types of OUTER JOIN.
- In a NATURAL JOIN on two relations R and S, no join condition is specified; an implicit EQUIJOIN condition for each pair of attributes with the same name from R and S is created.
- Each such pair of attributes is included only once in the resulting relation
- If the names of the join attributes are not the same in the base relations, it is possible to rename the attributes so that they match, and then to apply NATURAL JOIN. In this case, the AS construct can be used to rename a relation and all its attributes in the FROM clause

```
Q1B:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE NATURAL JOIN
                  (DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
        WHERE     Dname='Research';
```

Aggregate Functions in SQL

- Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary.
- Grouping is used to create subgroups of tuples before summarization.
- A number of built-in aggregate functions exist: COUNT, SUM, MAX, MIN, and AVG.
- The COUNT function returns the number of tuples or values as specified in a query.
- The functions SUM, MAX, MIN, and AVG can be applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values.
- These functions can be used in the SELECT clause or in a HAVING clause

Query 19. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
Q19:  SELECT  SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM    EMPLOYEE;
```

If we want to get the preceding function values for employees of a specific department—say, the 'Research' department—we can write Query 20, where the EMPLOYEE tuples are restricted by the WHERE clause to those employees who work for the 'Research' department.

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:  SELECT  SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM    (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE   Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:  SELECT  COUNT (*)
      FROM    EMPLOYEE;

Q22:  SELECT  COUNT (*)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNO=DNUMBER AND DNAME='Research';
```

Here the asterisk (*) refers to the *rows* (tuples), so COUNT (*) returns the number of rows in the result of the query. We may also use the COUNT function to count values in a column rather than tuples, as in the next example.

Query 23. Count the number of distinct salary values in the database.

```
Q23:  SELECT  COUNT (DISTINCT Salary)
      FROM    EMPLOYEE;
```

Grouping: The GROUP BY and HAVING Clauses

- In many cases we want to apply the aggregate functions to subgroups of tuples in a relation, where the subgroups are based on some attribute values. For example, we may want to find the average salary of employees in each department or the number of employees who work on each project.

- SQL has a GROUP BY clause for this purpose. The GROUP BY clause specifies the grouping attributes, which should also appear in the SELECT clause, so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).

Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q24:  SELECT    Dno, COUNT (*), AVG (Salary)
      FROM      EMPLOYEE
      GROUP BY  Dno;
```

Query 25. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
Q25:  SELECT    Pnumber, Pname, COUNT (*)
      FROM      PROJECT, WORKS_ON
      WHERE     Pnumber=Pno
      GROUP BY  Pnumber, Pname;
```

Query 26. For each project *on which more than two employees work*, retrieve the project number, the project name, and the number of employees who work on the project.

```
Q26:  SELECT     Pnumber, Pname, COUNT (*)
      FROM       PROJECT, WORKS_ON
      WHERE      Pnumber=Pno
      GROUP BY   Pnumber, Pname
      HAVING     COUNT (*) > 2;
```

Query 27. For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

```
Q27:  SELECT     Pnumber, Pname, COUNT (*)
      FROM       PROJECT, WORKS_ON, EMPLOYEE
      WHERE      Pnumber=Pno AND Ssn=Essn AND Dno=5
      GROUP BY   Pnumber, Pname;
```

Specifying General Constraints as Assertions in SQL

- In SQL, users can specify general constraints via declarative assertions, using the CREATE ASSERTION statement of the DDL.
- Each assertion is given a constraint name and is specified via a condition similar to the WHERE clause of an SQL query.

Assertions are different from check constraints in the way that check constraints are rules that relate to one single row only. Assertions, on the other hand, can involve any number of other tables, or any number of other rows in the same table. Assertions also check a condition, which must return a Boolean value. We can take an illustrative example.

Let us imagine that we have the following table, which contains employees in a company — we then also store an attribute containing their salary.

ID	Name	Age	DepNo	Salary
0	Hannah	18	10	1000$
1	Gavin	45	2	25.000$
2	Bobby	70	21	700$

We then want to make an assertion that there is no employee in our database, which is paid more than 30.000$ or less than 500$. It would then

```
Create Assertion SalaryAssertion check
                (Not exists
                        (select ID
                        From People p
                        Where p.salary > 30.000 OR p.salary < 500
);
```

Then it makes sure that we never have someone who receives a salary outside these bounds.

- For example, to specify the constraint that the salary of an employee must not be greater than the salary of the manager of the department that the employee works for in SQL, we can write the following assertion:

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS  ( SELECT     *
                      FROM       EMPLOYEE E, EMPLOYEE M,
                                 DEPARTMENT D
                      WHERE      E.Salary>M.Salary
                                 AND E.Dno=D.Dnumber
                                 AND D.Mgr_ssn=M.Ssn ) );
```

- The constraint name SALARY_CONSTRAINT is followed by the keyword CHECK, which is followed by a condition in parentheses that must hold true on every database state for the assertion to be satisfied. The constraint name can be used later to refer to the constraint or to modify or drop it. The DBMS is responsible for ensuring that the condition is not violated

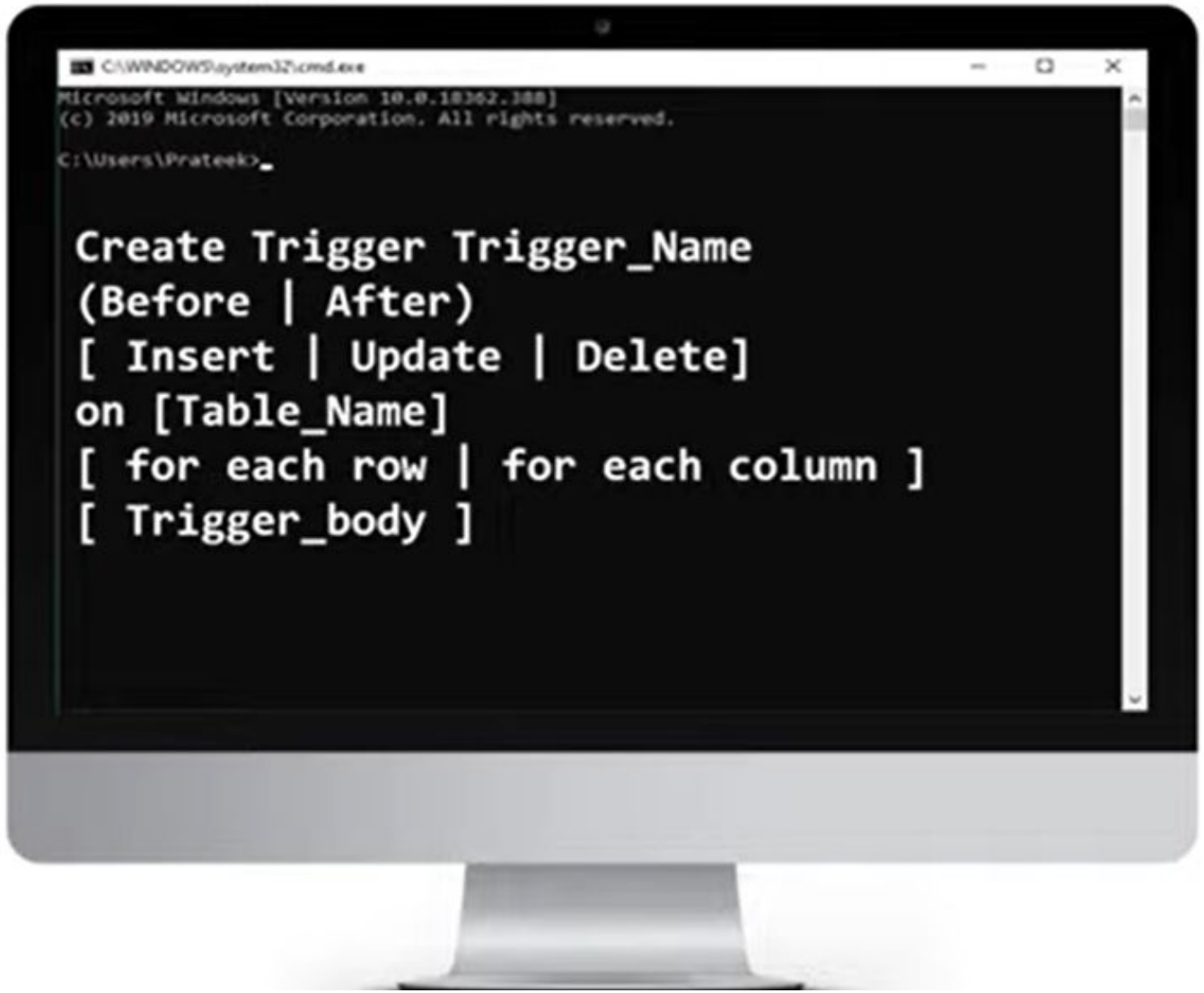
Introduction to Triggers in SQL

- A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs.
- Action to be taken when certain events occur and when certain conditions are satisfied.
- For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
- It may be useful to specify a condition that, if violated, causes some user to be informed of the violation

Benefits of Triggers

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

SYNTAX OF TRIGGER



Several events can trigger this rule: inserting a new employee record, changing an employee's salary, or changing an employee's supervisor. Suppose that the action to take would be to call an external stored procedure SALARY_VIOLATION, which will notify the supervisor.

R5: **CREATE TRIGGER** SALARY_VIOLATION

BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN **ON** EMPLOYEE

FOR EACH ROW

WHEN (**NEW**.SALARY > (**SELECT** SALARY **FROM** EMPLOYEE

WHERE SSN = **NEW**.SUPERVISOR_SSN))

INFORM_SUPERVISOR(**NEW**.Supervisor_ssn, **NEW**.Ssn);

The trigger is given the name SALARY_VIOLATION, which can be used to remove or deactivate the trigger later.

A typical trigger **has three components**:

1. The **event(s):** These are usually database update operations that are explicitly applied to the database. In this example the events are: inserting a new employee record, changing an employee's salary, or changing an employee's supervisor. In some cases, it may be necessary to write more than one trigger to cover all possible cases. These events are specified after the keyword **BEFORE** in our example, which means that the trigger should be executed before the triggering operation is executed. An alternative is to use the keyword **AFTER**, which specifies that the trigger should be executed after the operation specified in the event is completed.

2. The condition that determines whether the rule action should be executed: Once the triggering event has occurred, an optional condition may be evaluated. If no condition is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only if it evaluates to true will the rule action be executed. The condition is specified in the WHEN clause of the trigger.

3. The action to be taken: The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed. In this example, the action is to execute the stored procedure INFORM_SUPERVISOR.

Views (Virtual Tables) 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 necessarily exist in physical form;
  - it is considered to be a virtual table, in contrast to base tables,
  - whose tuples are always physically stored in the database.
- This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view.

- We can think of a view as a way of specifying a table that we need to reference frequently, even though it may not exist physically.
- Views, which are a type of virtual tables allow users to do the following
  - Structure data in a way that users or classes of users find natural or intuitive.
  - Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
  - Summarize data from various tables which can be used to generate reports.

- In SQL, the command to specify a view is CREATE VIEW.
- The view is given a (virtual) table name (or view name), a list of attribute names, and a query to specify the contents of the view.
- If none of the view attributes results from applying functions or arithmetic operations, we do not have to specify new attribute names for the view, since they would be the same as the names of the attributes of the defining tables in the default case.

- The views in V1 and V2 create virtual tables whose schemas are illustrated in Figure 5.2

```
V1:   CREATE VIEW   WORKS_ON1
      AS SELECT     Fname, Lname, Pname, Hours
          FROM      EMPLOYEE, PROJECT, WORKS_ON
          WHERE     Ssn=Essn AND Pno=Pnumber;

V2:   CREATE VIEW   DEPT_INFO(Dept_name, No_of_emps, Total_sal)
      AS SELECT     Dname, COUNT (*), SUM (Salary)
          FROM      DEPARTMENT, EMPLOYEE
          WHERE     Dnumber=Dno
          GROUP BY  Dname;
```

Figure 5.2
Two views specified on the database schema of Figure 3.5.

WORKS_ON1

Fname	Lname	Pname	Hours

DEPT_INFO

Dept_name	No_of_emps	Total_sal

We can now specify SQL queries on a view—or virtual table—in the same way we specify queries involving base tables. For example, to retrieve the last name and first name of all employees who work on the 'ProductX' project, we can utilize the WORKS_ON1 view and specify the query as in QV1:

```
QV1:    SELECT    Fname, Lname
        FROM      WORKS_ON1
        WHERE     Pname='ProductX';
```

If we do not need a view any more, we can use the **DROP VIEW** command to dispose of it. For example, to get rid of the view V1, we can use the SQL statement in V1A:

```
V1A:    DROP VIEW    WORKS_ON1;
```

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

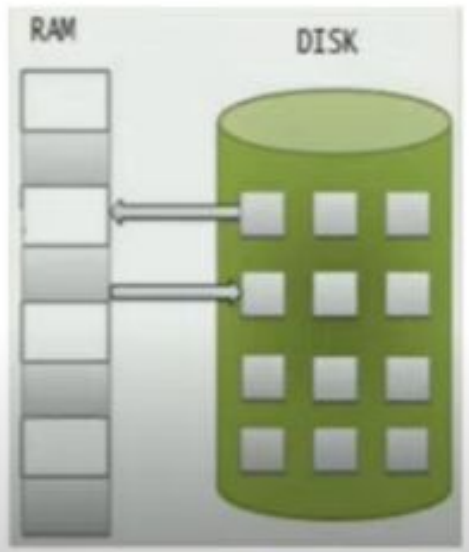
- Logical files do not contain data.
- They contain a description of records found in one or more physical files.
- A logical file is a view or representation of one or more physical files.
- Logical files that contain more than one format are referred to as multi-format logical files.

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.

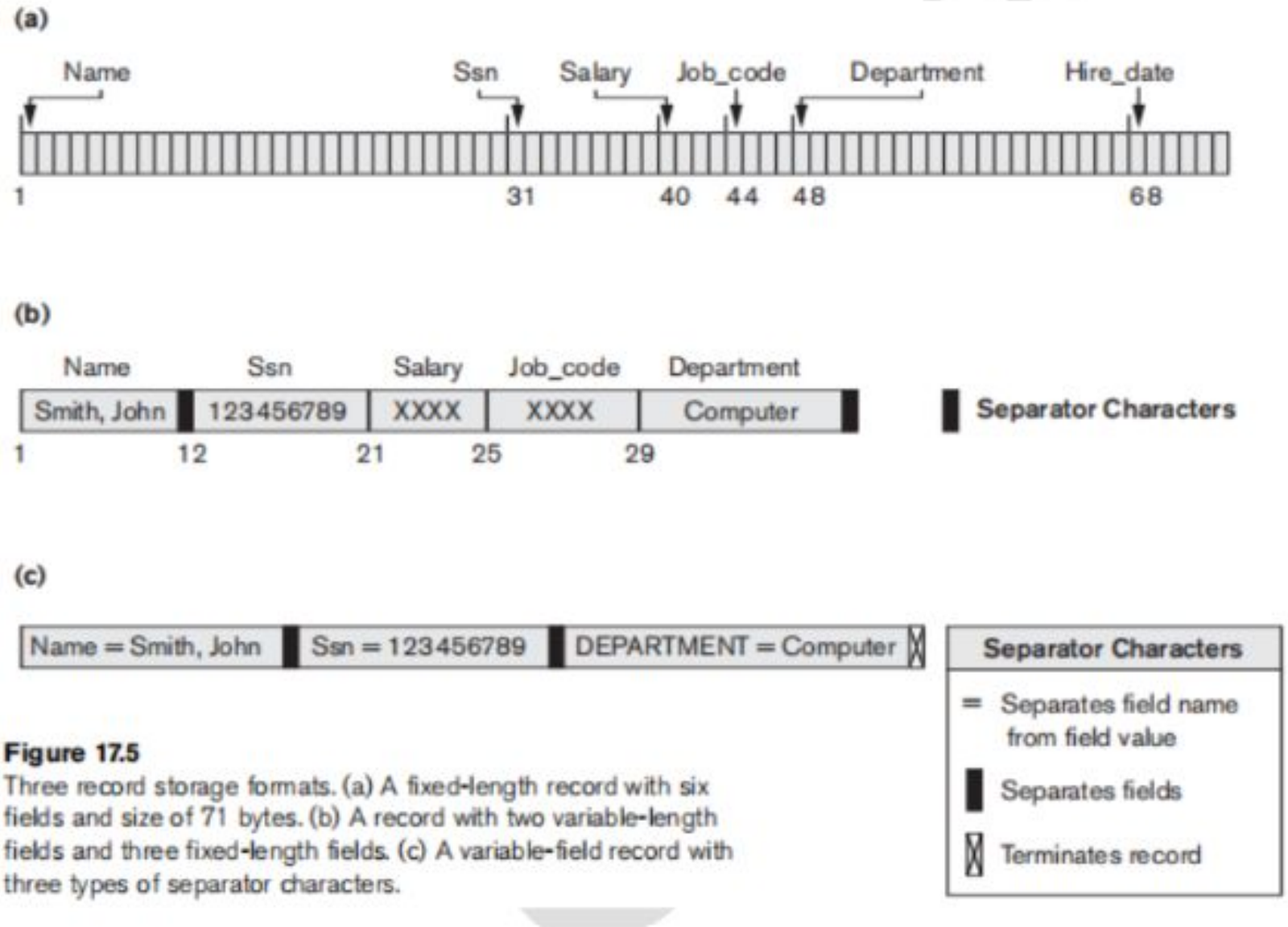


Figure 17.5
Three record storage formats. (a) A fixed-length record with six fields and size of 71 bytes. (b) A record with two variable-length fields and three fixed-length fields. (c) A variable-field record with three types of separator characters.

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

SPANNED ORGANIZATION

Record (150 KB)

Student Table

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

UNSPANNED ORGANIZATION

Record (150 KB)

Student Table

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

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 ≥ R,
- Then no. of records per block /blocking factor
  bfr = ⌊B/R⌋

where the ⌊(x)⌋ (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 = ⌈(r/bfr)⌉ blocks

where the ⌈(x)⌉ (ceiling function) rounds the value x up to the next integer

for Spanned organization

number of blocks

= ⌈no. of records X recordsize / blocksize ⌉

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

Files of Unordered Records (Heap Files)

- records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file.
- This organization is often used with additional access paths, such as the secondary indexes
- Insertion
  - Inserting a new record is very efficient.
  - The last disk block of the file is copied into a buffer, the new record is added, and the block is then rewritten back to disk.
  - The address of the last file block is kept in the file header.

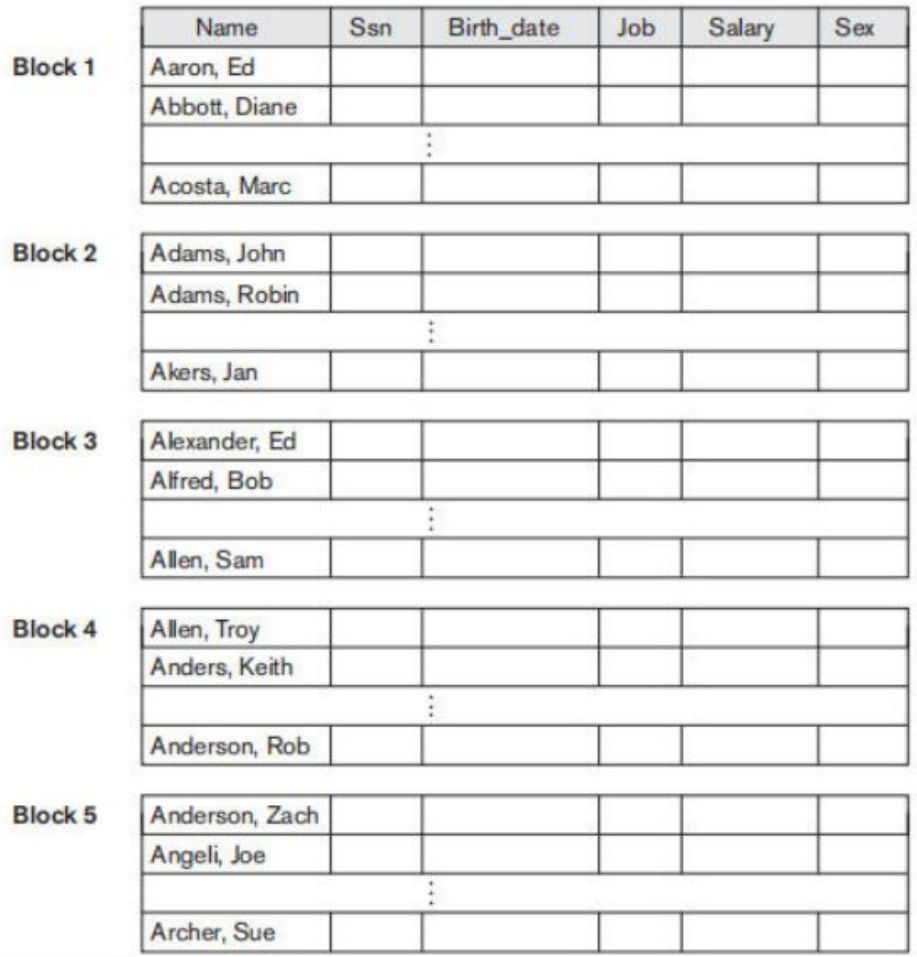
- Searching
  - searching a record using any search condition involves a linear search through the file block by block an expensive procedure.
  - If only one record satisfies the search condition, then, on the average, a program will read into memory and search half the file blocks before it finds the record.
  - For a file of b blocks, this requires searching (b/2) blocks, on average.
  - If no records or several records satisfy the search condition, the program must read and search all b blocks in the file

- Deletion
  - a program must first find its block, copy the block into a buffer, delete the record from the buffer, and finally rewrite the block back to the disk.
  - This leaves unused space in the disk block.
  - Deleting a large number of records in this way results in wasted storage space.
  - Another technique used for record deletion is to have an extra byte or bit, called a deletion marker, stored with each record.
  - A record is deleted by setting the deletion marker to a certain value. A different value for the marker indicates a valid (not deleted) record. Search programs consider only valid records in a block when conducting their search.
  - Both of these deletion techniques require periodic reorganization of the file to reclaim the unused space of deleted records

Files of Ordered Records (Sorted Files)

- We can physically order the records of a file on disk based on the values of one of their fields called the ordering field.
- This leads to an ordered or sequential file.
- If the ordering field is also a key field of the file a field guaranteed to have a unique value in each record then the field is called the ordering key for the file.

Figure 17.7
Some blocks of an ordered (sequential) file of EMPLOYEE records with Name as the ordering key field.



	Name	Ssn	Birth_date	Job	Salary	Sex
Block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
Block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
Block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
Block 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
Block 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					

Average Access Times

- The following table shows the average access time to access a specific record for a given type of file

TABLE 13.2 AVERAGE ACCESS TIMES FOR BASIC FILE ORGANIZATIONS

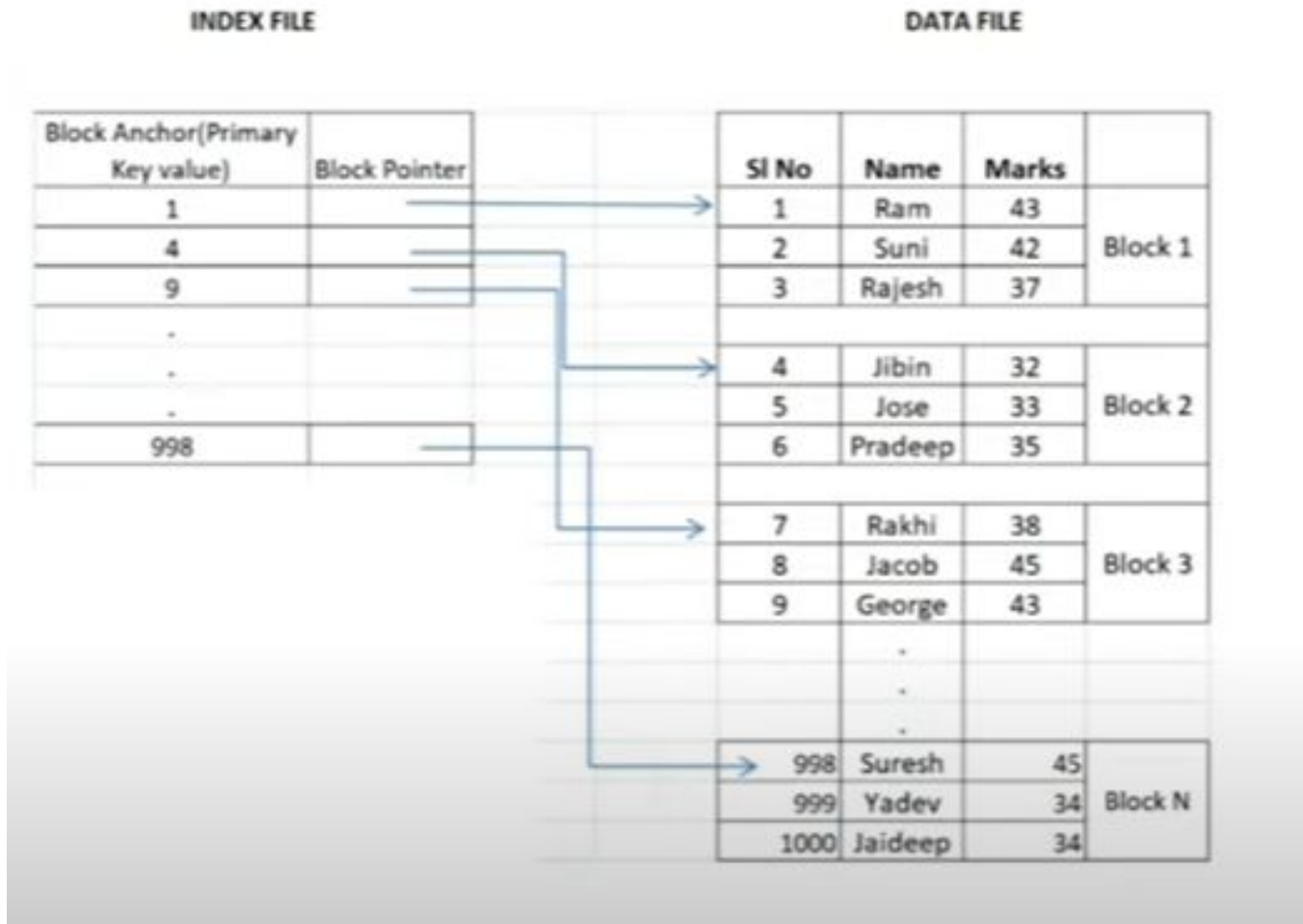
TYPE OF ORGANIZATION	ACCESS/SEARCH METHOD	AVERAGE TIME TO ACCESS A SPECIFIC RECORD
Heap (Unordered)	Sequential scan (Linear Search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary Search	$\log_2 b$

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.

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

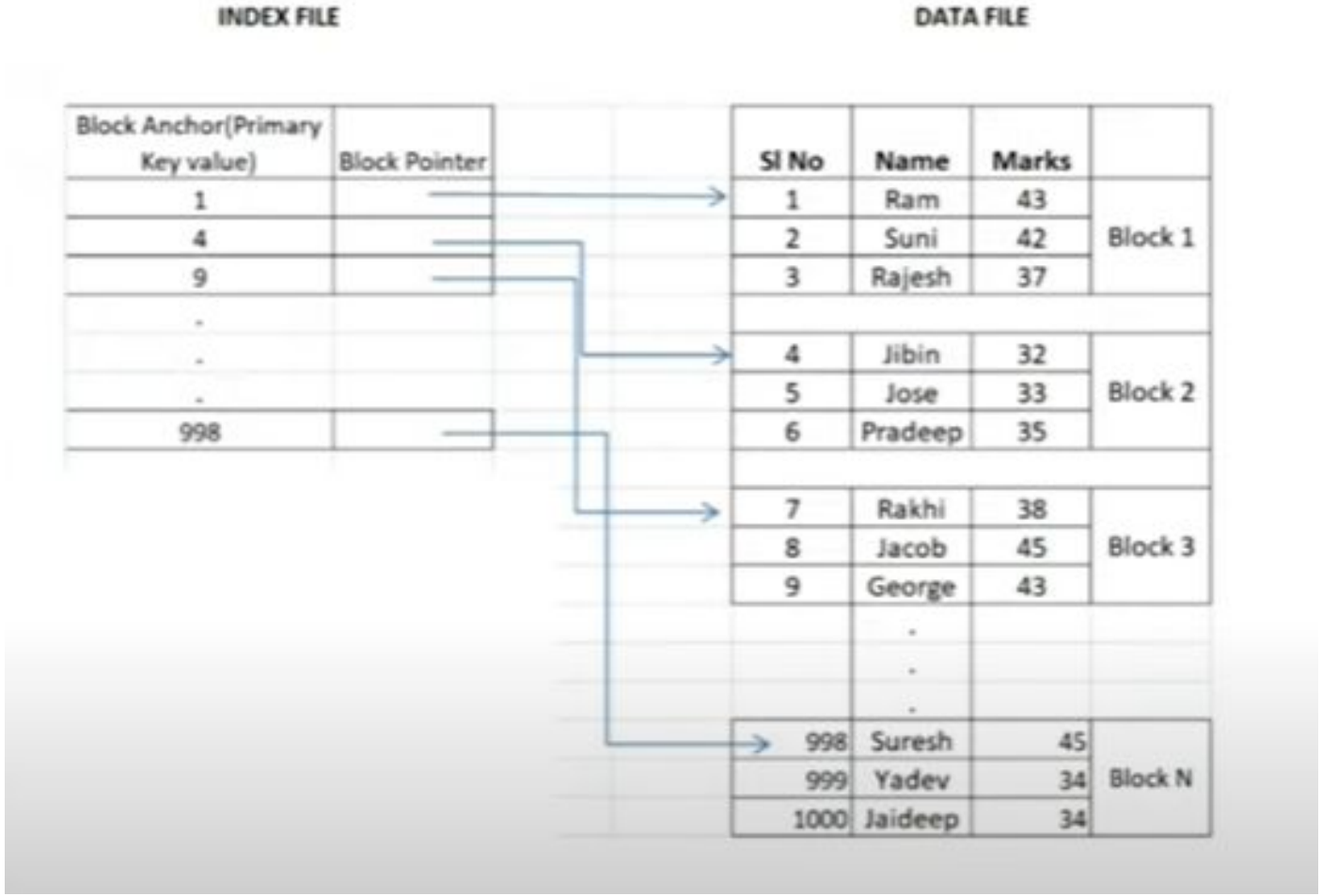

INDEX FILE
DATA FILE
Block Anchor(Primary Key value)
Block Pointer
1
4
9
998
Sl No
Name
Marks
1 Ram 43
2 Suni 42
3 Rajesh 37
Block 1
4 Jibin 32
5 Jose 33
6 Pradeep 35
Block 2
7 Rakhi 38
8 Jacob 45
9 George 43
Block 3
998 Suresh 45
999 Yadev 34
1000 Jaideep 34
Block N

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

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.

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

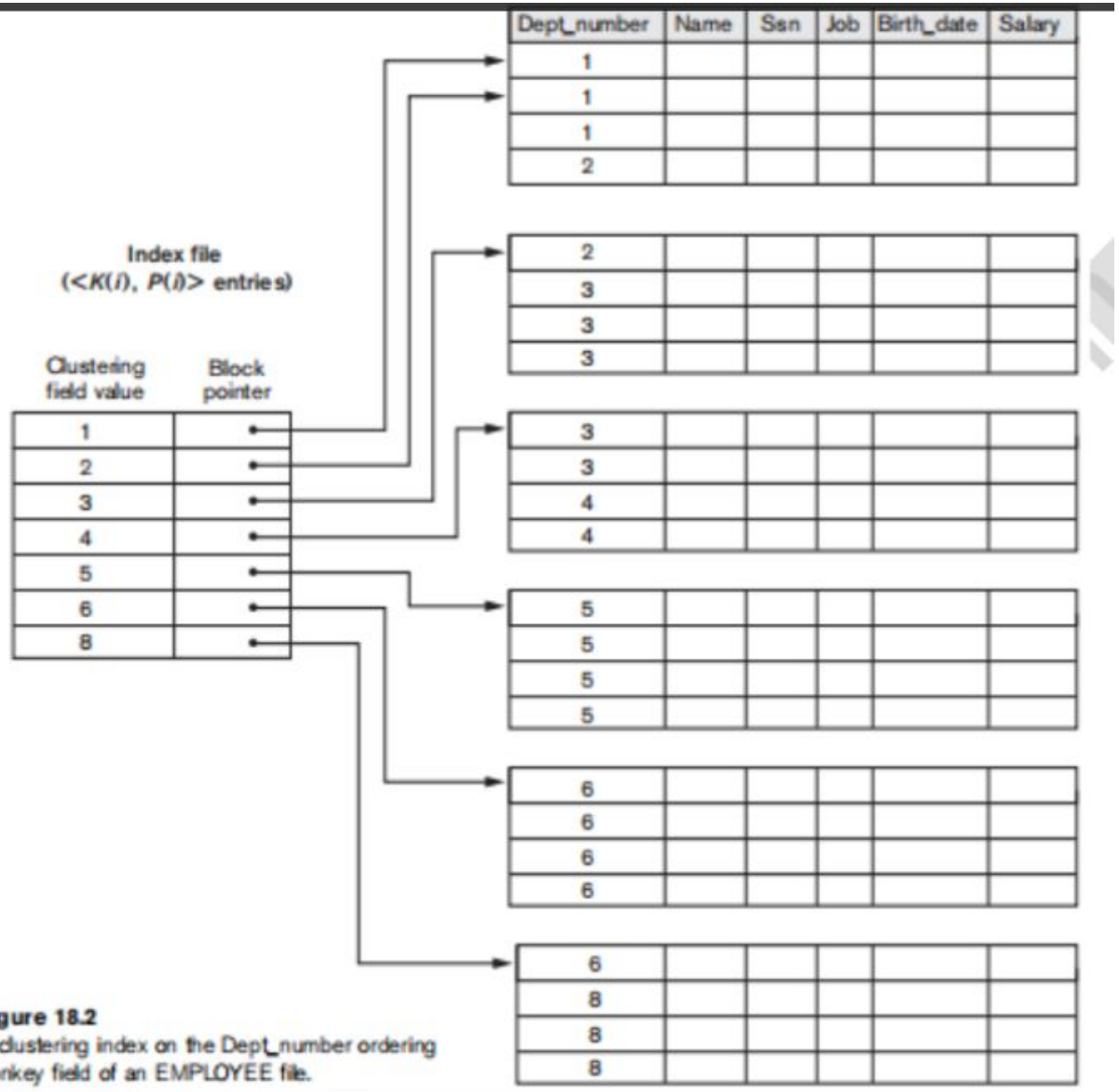
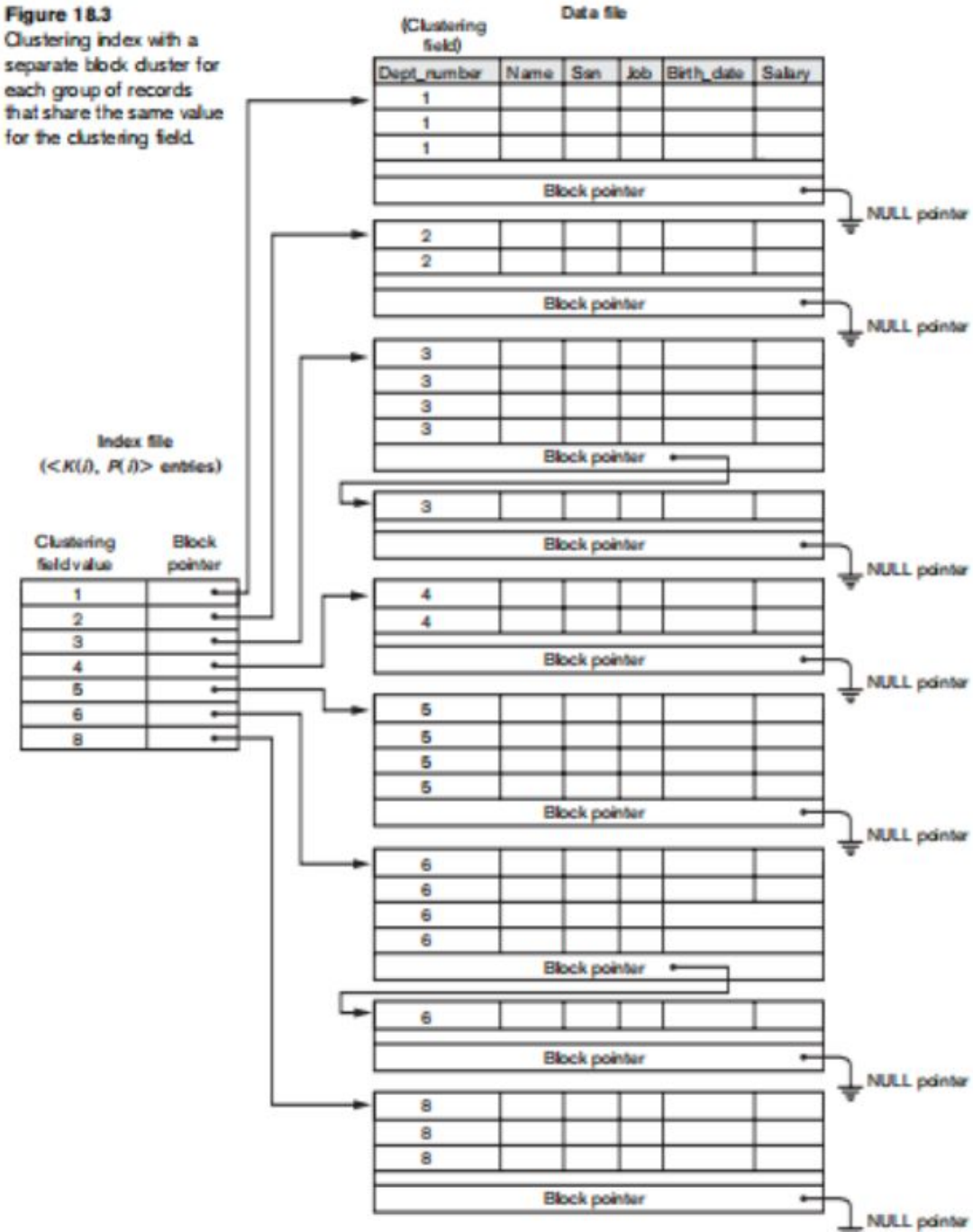


Figure 18.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

CLUSTERING INDEX

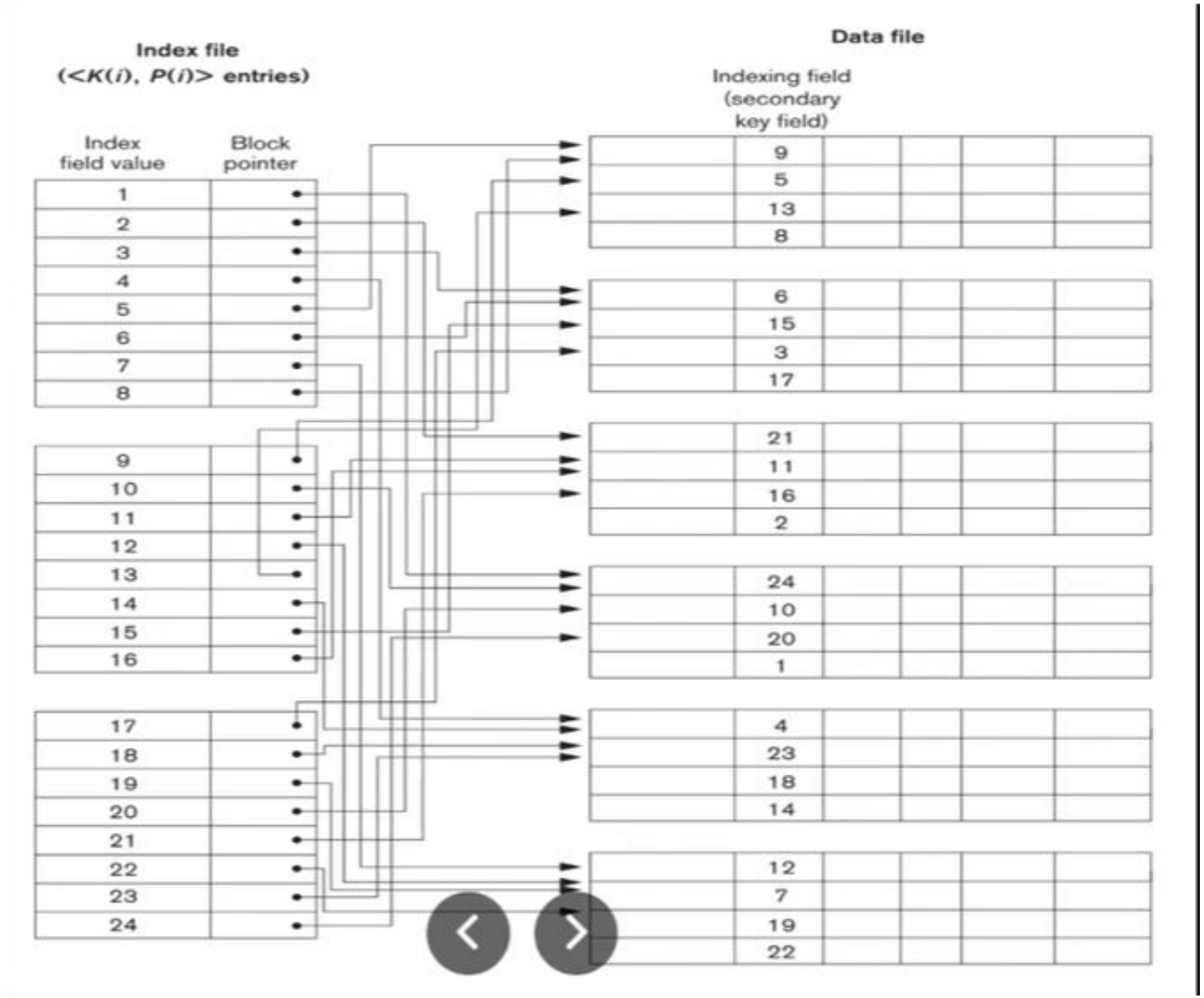
- A clustering index is another example of a **nondense index**, because it has an entry for every distinct value of the indexing field
- Here record insertion and deletion still cause problems, because the data records are physically ordered.
- To avoid the problem of insertion, it is common to **reserve 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.

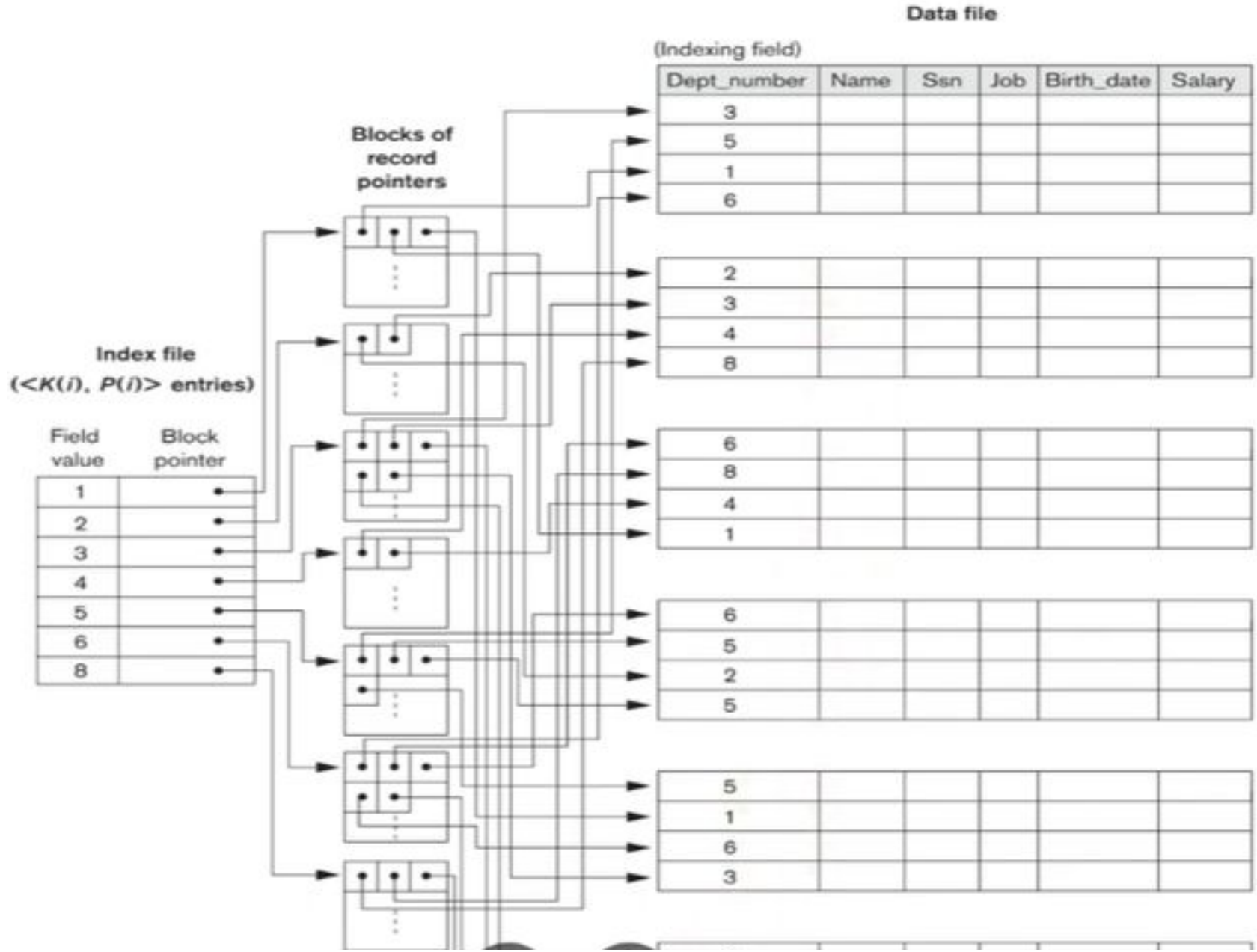


SECONDARY INDEX

- 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

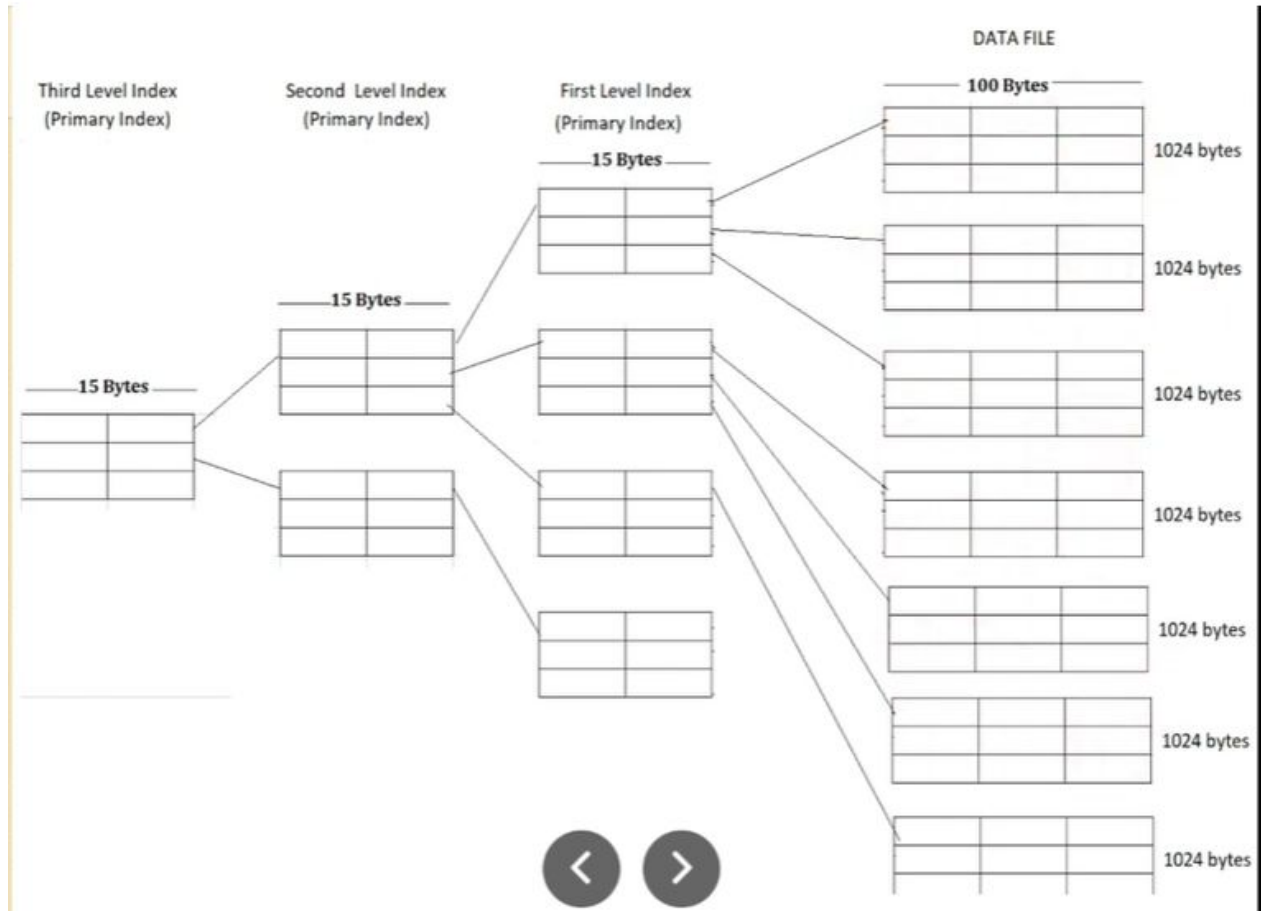

Index file
(<K(i), P(i)> entries)
Index field value
Block pointer
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Data file
Indexing field (secondary key field)
9
5
13
8
6
15
3
17
21
11
16
2
24
10
20
1
4
23
18
14
12
7
19
22

- 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

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.


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

- 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.