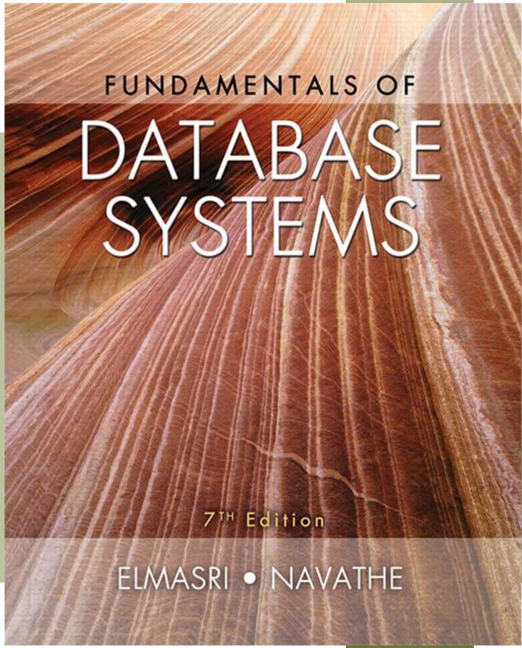
Data Base Management System

Module –III



By

Dr. Jagadamba G

Dept. of ISE, SIT, Tumakuru

Schema Definition, Basic Constraints, and Queries

Data Definition, Constraints, and Schema Changes

- Used to CREATE, DROP, and ALTER the descriptions of the tables (relations) of a database

CREATE TABLE

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- A constraint NOT NULL may be specified on an attribute

-
  ```
  CREATE TABLE   DEPARTMENT
  (DNAME           VARCHAR(10)  NOT NULL,
  DNUMBER          INTEGER       NOT NULL,
  MGRSSN           CHAR(9),
  MGRSTARTDATE   CHAR(9)  );
  ```

CREATE TABLE

- In SQL2, can use the CREATE TABLE command for specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys).
- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases

```
CREATE TABLE   DEPT
(     DNAME   VARCHAR(10)       NOT NULL,
      DNUMBER              INTEGERNOT NULL,
      MGRSSN               CHAR(9),
      MGRSTARTDATE  CHAR(9),
      PRIMARY KEY (DNUMBER),
      UNIQUE (DNAME),
      FOREIGN KEY (MGRSSN) REFERENCES EMP  );
```

CREATE AND DROP SCHEMA

- Specifies a new database schema by giving it a name
- The CREATE DATABASE statement is used to create a new SQL database.
- CREATE DATABASE databasename;
- The DROP DATABASE statement is used to drop an existing SQL database.
- DROP DATABASE databasename;

REFERENTIAL INTEGRITY OPTIONS

- We can specify RESTRICT, CASCADE, SET NULL or SET DEFAULT on referential integrity constraints (foreign keys)

```
CREATE TABLE   DEPT
 (      DNAME           VARCHAR(10)  NOT NULL,
        DNUMBER         INTEGER      NOT NULL,
        MGRSSN          CHAR(9),
        MGRSTARTDATE         CHAR(9),
        PRIMARY KEY (DNUMBER),
        UNIQUE (DNAME),
        FOREIGN KEY (MGRSSN) REFERENCES EMP
ON DELETE SET DEFAULT ON UPDATE CASCADE  );
```

REFERENTIAL INTEGRITY OPTIONS (continued)

```
CREATE TABLE   EMP
    (           ENAME   VARCHAR(30)      NOT NULL,
                ESSN    CHAR(9),
                BDATE   DATE,
                DNO     INTEGER  DEFAULT 1,
                SUPERSSN            CHAR(9),
                PRIMARY KEY (ESSN),
                FOREIGN KEY (DNO) REFERENCES DEPT
        ON DELETE SET DEFAULT ON UPDATE CASCADE,
                FOREIGN KEY (SUPERSSN) REFERENCES EMP
        ON DELETE SET NULL ON UPDATE CASCADE  );
```

Data Types in SQL

Has DATE, TIME, and TIMESTAMP data types

- **DATE:**
  - Made up of year-month-day in the format yyyy-mm-dd
- **TIME:**
  - Made up of hour:minute:second in the format hh:mm:ss
- **TIME(i):**
  - Made up of hour:minute:second plus i additional digits specifying fractions of a second
  - format is hh:mm:ss:ii...i
- **TIMESTAMP:**
  - Has both DATE and TIME components

- **INTERVAL:**
  - Specifies a relative value rather than an absolute value
  - Can be DAY/TIME intervals or YEAR/MONTH intervals
  - Can be positive or negative when added to or subtracted from an absolute value, the result is an absolute value

DROP TABLE

- Used to remove a relation (base table) *and its definition*
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

```
DROP TABLE  DEPENDENT;
```

ALTER TABLE

The possible alter table actions include

- Adding or dropping a column (attribute)
- Changing a column definition
- Adding or dropping table constraints
- Add an attribute to one of the base relations

The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute.

ALTER TABLE

- Example:to add an attribute for keeping track of jobs of employees to the EMPLOYEE base relation in the COMPANY schema, we can use the command

```
ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);
```

OR

```
ALTER TABLE  EMPLOYEE  ADD   JOB   VARCHAR(12);
```

- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

ALTER TABLE

- To drop a column, we must choose either CASCADE or RESTRICT for drop behavior.
- If CASCADE is chosen, all constraints and views that reference the column are dropped automatically from the schema, along with the column.

ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;

- One can also change the constraints specified on a table by adding or dropping a named constraint.
- To be dropped, a constraint must have been given a name when it was specified.
- For example, to drop the constraint named EMPSUPERFK in the EMPLOYEE relation, we write:

ALTER TABLE COMPANY.EMPLOYEE

DROP CONSTRAINT EMPSUPERFK CASCADE;

ALTER TABLE

- For example, to drop the constraint named EMPSUPERFK in the EMPLOYEE relation, we write:

ALTER TABLE COMPANY.EMPLOYEE

DROP CONSTRAINT EMPSUPERFK CASCADE;

- Once the above statement is executed, we can redefine a replacement constraint by adding a new constraint to the relation, if needed.
- This is specified by using the **ADD CONSTRAINT** keyword in the ALTER TABLE statement followed by the new constraint, which can be named or unnamed and can be of any of the table constraint types discussed.

ALTER TABLE

- Rename can be done for column name

```
ALTER TABLE table_name
RENAME COLUMN old-name TO new_name;
EX:  ALTER TABLE Employee
       RENAME COLUMN eid TO employeeid;
```

- Modify datatype of coumn

```
ALTER TABLE table_name
ALTER COLUMN columnname datatype;
Ex: ALTER TABLE Employee
      ALTER COLUMN dataofbirth date;
```

Specifying Updates in SQL

- There are three SQL commands to modify the database; INSERT, DELETE, and UPDATE

INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

INSERT (cont.)

- Example:

```
U1:   INSERT INTO  EMPLOYEE
      VALUES ('Richard','K','Marini', '653298653', '30-DEC-52',
      '98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4 )
```

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
- Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
U1A:   INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
        VALUES ('Richard', 'Marini', '653298653')
```

INSERT (cont.)

- Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database
- Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation

INSERT (cont.)

– Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table DEPTS_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

```
U3A:      CREATE TABLE  DEPTS_INFO
                   (DEPT_NAME        VARCHAR(10),
                    NO_OF_EMPS       INTEGER,
                    TOTAL_SAL        INTEGER);

U3B:      INSERT INTO        DEPTS_INFO (DEPT_NAME,
                   NO_OF_EMPS, TOTAL_SAL)
          SELECT             DNAME, COUNT (*), SUM (SALARY)
          FROM               DEPARTMENT, EMPLOYEE
          WHERE              DNUMBER=DNO
          GROUP BY           DNAME ;
```

INSERT (cont.)

- Note: The DEPTS_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations *after* issuing U3B. We have to create a view (see later) to keep such a table up to date.

DELETE

- Removes tuples from a relation
- Includes a WHERE-clause to select the tuples to be deleted
- Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
- Referential integrity should be enforced

DELETE (cont.)

- Examples:

```
U4A:    DELETE FROM      EMPLOYEE
        WHERE            LNAME='Brown’

U4B:    DELETE FROM      EMPLOYEE
        WHERE            SSN='123456789’

U4C:    DELETE FROM      EMPLOYEE
        WHERE            DNO  IN
              (SELECT    DNUMBER
              FROM  DEPARTMENT
              WHERE      DNAME='Research')

U4D:    DELETE FROM      EMPLOYEE
```

UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

UPDATE (cont.)

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5: UPDATE      PROJECT
    SET         PLOCATION = 'Bellaire', DNUM = 5
    WHERE       PNUMBER=10
```

UPDATE (cont.)

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6:  UPDATE  EMPLOYEE
     SET     SALARY = SALARY *1.1
     WHERE   DNO  IN (SELECT  DNUMBER
                      FROM    DEPARTMENT
                      WHERE DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
- The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
- The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the SELECT statement
- This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model; SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
- Hence, an SQL relation (table) is a *multi-set* (sometimes called a bag) of tuples; it *is not* a set of tuples
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

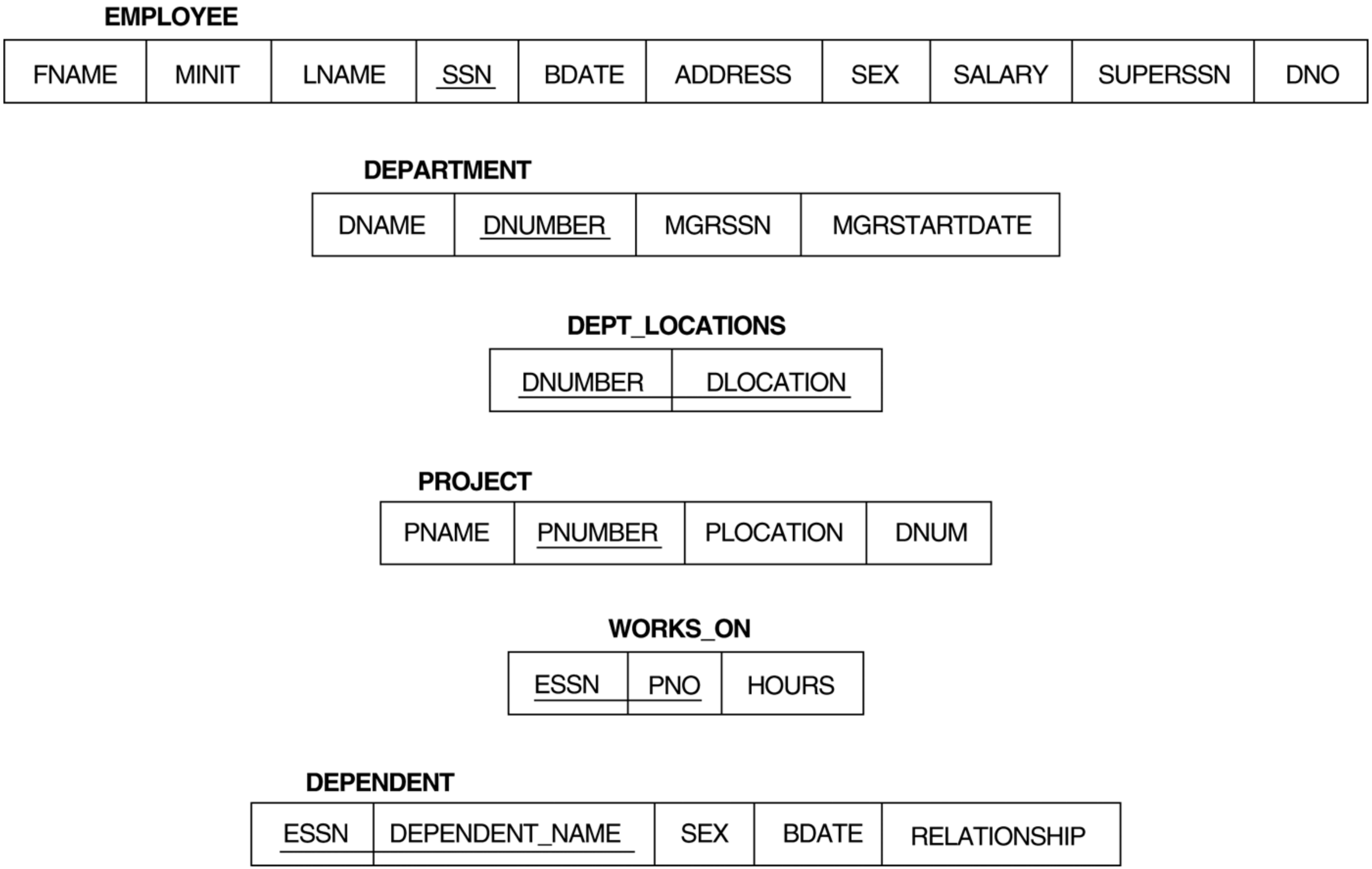
Retrieval Queries in SQL (cont.)

- Basic form of the SQL SELECT statement is called a *mapping* or a *SELECT-FROM-WHERE block*

```
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

Relational Database Schema--Figure 5.5



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

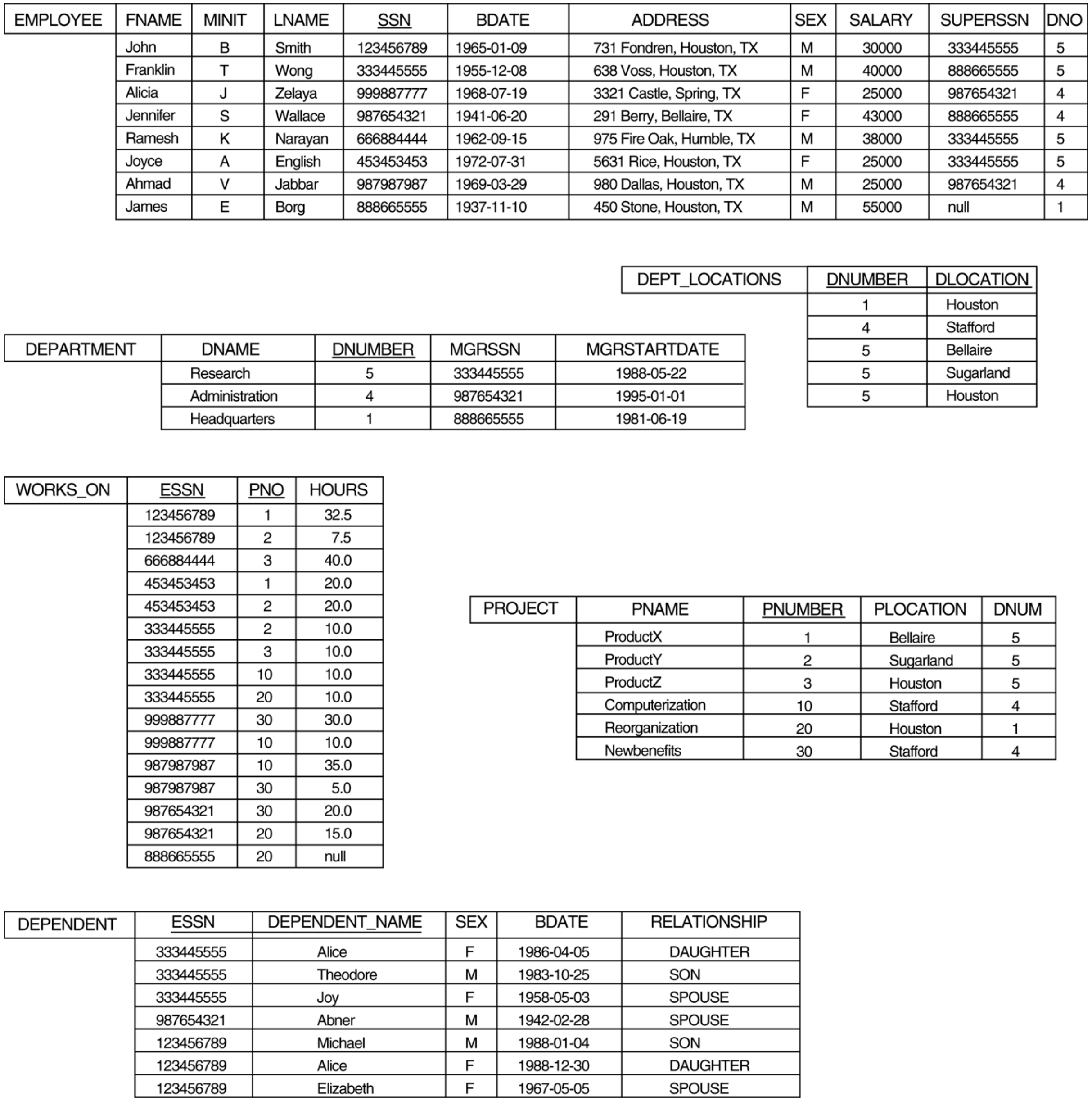
WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Populated Database--Fig.5.6



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

Simple SQL Queries

- Basic SQL queries correspond to using the SELECT, PROJECT, and JOIN operations of the relational algebra
- All subsequent examples use the COMPANY database
- Example of a simple query on *one* relation
- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

```
Q0:  SELECT    BDATE, ADDRESS
     FROM      EMPLOYEE
     WHERE     FNAME='John' AND MINIT='B'  AND LNAME='Smith'
```

  - Similar to a SELECT-PROJECT pair of relational algebra operations; the SELECT-clause specifies the *projection attributes* and the WHERE-clause specifies the *selection condition*
  - However, the result of the query *may contain* duplicate tuples

Simple SQL Queries (cont.)

- 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
```

  - Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
  - (DNAME='Research') is a *selection condition* (corresponds to a SELECT operation in relational algebra)
  - (DNUMBER=DNO) is a *join condition* (corresponds to a JOIN operation in relational algebra)

Simple SQL Queries (cont.)

- 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 birthdate.

```
Q2: SELECT   PNUMBER, DNUM, LNAME, BDATE, ADDRESS
    FROM     PROJECT, DEPARTMENT, EMPLOYEE
    WHERE    DNUM=DNUMBER AND MGRSSN=SSN  AND
    PLOCATION='Stafford'
```

  - In Q2, there are *two* join conditions
  - The join condition DNUM=DNUMBER relates a project to its controlling department
  - The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*
- A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name

SYNATX: When alias is used on column:

```
SELECT column_name AS alias_name
FROM table_name;
```

SYNATX: When alias is used on table:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Example:

- EMPLOYEE.LNAME, DEPARTMENT.DNAME
- **Query: SELECT EMPLOYEEID AS ID**
  FROM EMPLOYEE;

ALIASES

- Some queries need to refer to the same relation twice
- In this case, *aliases* are given to the relation name
- Query 8: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
Q8:  SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
     FROM     EMPLOYEE E S
     WHERE    E.SUPERSSN=S.SSN
```

  - In Q8, the alternate relation names E and S are called *aliases* or *tuple variables* for the EMPLOYEE relation
  - We can think of E and S as two *different copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*

ALIASES (cont.)

- Aliasing can also be used in any SQL query for convenience
  Can also use the AS keyword to specify aliases

```
Q8:      SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
         FROM     EMPLOYEE AS E, EMPLOYEE AS S
         WHERE    E.SUPERSSN=S.SSN
```

UNSPECIFIED WHERE-clause

- A *missing WHERE-clause* indicates no condition; hence, *all tuples* of the relations in the FROM-clause are selected
- This is equivalent to the condition WHERE TRUE
- <u>Query 9:</u> Retrieve the SSN values for all employees.

```
Q9:SELECT SSN
   FROM  EMPLOYEE
```

- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected

UNSPECIFIED WHERE-clause (cont.)

- Example:

```
Q10:        SELECT SSN, DNAME
            FROM   EMPLOYEE, DEPARTMENT
```

  - It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result

USE OF *

- To retrieve all the attribute values of the selected tuples, a * is used, which stands for *all the attributes*

  Examples:

```
Q1C:      SELECT      *
          FROM  EMPLOYEE
          WHERE       DNO=5

Q1D:      SELECT *
          FROM  EMPLOYEE, DEPARTMENT
          WHERE       DNAME='Research' AND
                      DNO=DNUMBER
```

USE OF DISTINCT

- SQL does not treat a relation as a set; *duplicate tuples can appear*
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:     SELECT SALARY
         FROM  EMPLOYEE

Q11A:    SELECT DISTINCT SALARY
         FROM  EMPLOYEE
```

Use of AND, OR, BETWEEN, IN and NOT IN Operators

- SELECT * FROM table_name WHERE condition1 AND condition2 AND…conditionN;
- SELECT * FROM table_name WHERE condition1 OR condition2 OR… conditionN;
- SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;
- SELECT column_name(s) FROM table_name WHERE column_name IN (list_of_values);
- SELECT column_name(s) FROM table_name WHERE Name LIKE '_pattern%';

Use of binary (=,<, > and <>) Operators

- SELECT * FROM table_name WHERE condition = value;
- SELECT * FROM table_name WHERE condition <> values;

ARITHMETIC OPERATIONS

- The standard arithmetic operators '+', '-'. '*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result
- Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q27: SELECT   FNAME, LNAME, 1.1*SALARY
     FROM     EMPLOYEE, WORKS_ON, PROJECT
     WHERE    SSN=ESSN AND PNO=PNUMBER AND  PNAME='ProductX’
```

SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a union operation (**UNION)**, and in *some versions* of SQL there are set difference (**MINUS)** and intersection (**INTERSECT)** operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations* ; the two relations must have the same attributes and the attributes must appear in the same order

- There are certain rules which must be followed to perform operations using SET operators in SQL. Rules are as follows:
  - ➢The number and order of columns must be the same.
  - ➢Data types must be compatible.

SET OPERATIONS -UNION

- The UNION operator is used to combine the result-set of two or more SELECT statements.
- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order
- **UNION Syntax:**

```
SELECT column_name(s) FROM table1
 UNION
 SELECT column_name(s) FROM table2;
```

- **UNION ALL Syntax**
- The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

SET OPERATIONS -UNION

- Query 4a: Make a list of cities (only distinct values) from both the "Customers" and the "Suppliers" table:

-
  ```
  SELECT City FROM Customers
  UNION
  SELECT City FROM Suppliers
  ORDER BY City;
  ```

City
Bangalore
Delhi
Finland
Trivandrum
Tumakuru

SET OPERATIONS -UNION

- Query 4b: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.
-

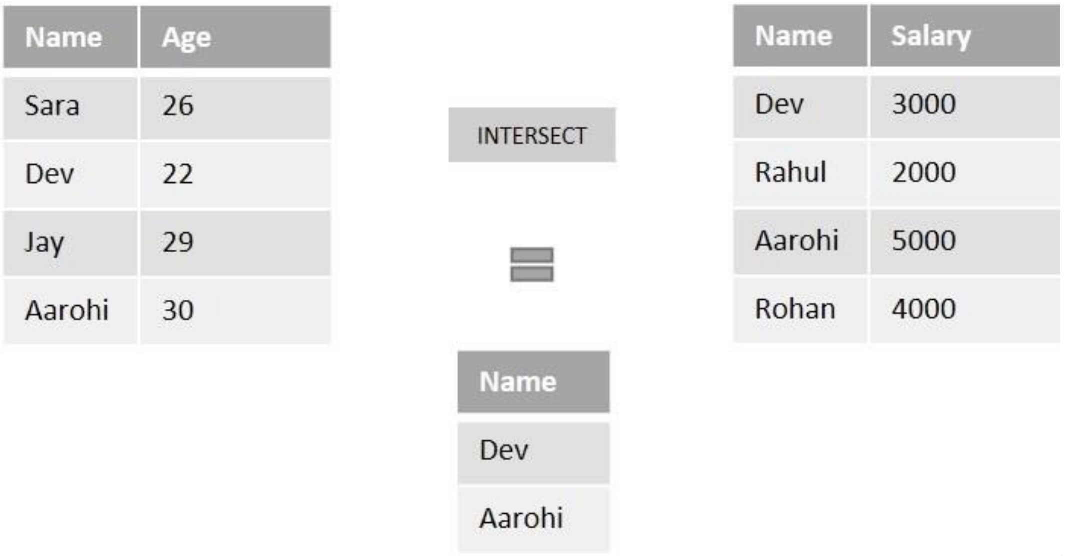
```
Q4:   (SELECT   PNAME
       FROM     PROJECT, DEPARTMENT, EMPLOYEE
       WHERE    DNUM=DNUMBER AND MGRSSN=SSN  AND LNAME='Smith')
       UNION
       (SELECT  PNAME
       FROM     PROJECT, WORKS_ON, EMPLOYEE
       WHERE    PNUMBER=PNO AND ESSN=SSN AND LNAME='Smith');
```

SET OPERATIONS -INTERSECTION

- The INTERSECT clause in SQL is used to combine two SELECT statements but the dataset returned by the INTERSECT statement will be the intersection of the data sets of the two SELECT statements.
- In simple words, the INTERSECT statement will return only those rows which will be common to both of the SELECT statements.

```
SELECT column1 [, column2 ]  FROM table1 [, table2 ] [WHERE condition]
INTERSECT
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
```

```
SELECT Name
FROM Customers
INTERSECT
SELECT name
FROM Salesman;
```



Name	Age
Sara	26
Dev	22
Jay	29
Aarohi	30

INTERSECT

Name	Salary
Dev	3000
Rahul	2000
Aarohi	5000
Rohan	4000

=

Name
Dev
Aarohi

SET OPERATIONS -INTERSECTION

```
SELECT NAME, AGE, HOBBY FROM STUDENTS_HOBBY
WHERE AGE BETWEEN 25 AND 30
INTERSECT
SELECT NAME, AGE, HOBBY FROM STUDENTS
WHERE AGE BETWEEN 20 AND 30;
```

Analyze the above query

```
NAME    AGE     HOBBY
Varun   26      Football
```

MINUS compares the data between tables and returns the rows of data that exist only in the first table you specify.

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

EXCEPT Operator

- The EXCEPT operator in SQL is used to retrieve all the unique records from the left operand (query), except the records that are present in the result set of the right operand (query).
- This operator compares the distinct values of the left query with the result set of the right query.
- If a value from the left query is found in the result set of the right query, it is excluded from the final result.

```
SELECT column1, column2,..., columnN
FROM table1, table2,..., tableN  [Conditions] //optional
EXCEPT
SELECT column1, column2,..., columnN
FROM table1, table2,..., tableN [Conditions] //optional

SELECT NAME, HOBBY, AGE FROM STUDENTS
EXCEPT
SELECT NAME, HOBBY, AGE FROM STUDENTS_HOBBY;
```

NESTING OF QUERIES

- **NESTED query/ INNER query/SUB query** is a SQL query within another SQL query embedded within a WHERE clause.
- The result of inner query is used in execution of outer query.
- Query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.
- Many of the previous queries can be specified in an alternative form using nesting.
- Query 11a: Write the employee name who is drawing the second highest salary.

```
SELECT EName
FROM Employee
WHERE IN  (SELECT MAX (Salary)
            FROM employee)
```

NESTING OF QUERIES

Query 11b: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q11:    SELECT  FNAME, LNAME, ADDRESS
        FROM    EMPLOYEE
        WHERE  DNO IN  (SELECT  DNUMBER
                        FROM  DEPARTMENT
                        WHERE  DNAME='Research' )
```

------------------------------OR (SQL without nesting)--

```
Q11b: SELECT E.FNAME,E.ADDRESS
      FROM EMPLOYEE E, DEPARTMENT D
      WHERE D.DNAME="RESEARCH" AND D.DNUMBER = E.DNO;
```

NESTING OF QUERIES (cont.)

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator **IN** compares a value v with a set (or multi-set) of values V, and evaluates to **TRUE** if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In this example, the nested query is *not correlated* with the outer query

CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be ***correlated***
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12: SELECT       E.FNAME, E.LNAME
     FROM         EMPLOYEE AS E
     WHERE        E.SSN IN (SELECT   ESSN
                            FROM     DEPENDENT
                            WHERE    ESSN=E.SSN AND E.FNAME=DEPENDENT_NAME)
```

CORRELATED NESTED QUERIES (cont.)

- In Q12, the nested query has a different result *for each tuple* in the outer query
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query. For example, Q12 may be written as in Q12A

```
Q12A:SELECT    E.FNAME, E.LNAME
     FROM      EMPLOYEE E, DEPENDENT D
     WHERE     E.SSN=D.ESSN AND E.FNAME=D.DEPENDENT_NAME
```

- The original SQL as specified for SYSTEM R also had a **CONTAINS** comparison operator, which is used in conjunction with nested correlated queries
- This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently

CORRELATED NESTED QUERIES (cont.)

- Most implementations of SQL *do not* have this operator
- The CONTAINS operator compares two *sets of values* , and returns TRUE if one set contains all values in the other set (reminiscent of the *division* operation of algebra).

Query 3: Retrieve the name of each employee who works on *all* the projects controlled by department number 5.

```
Q3:   SELECT    FNAME, LNAME
      FROM      EMPLOYEE
      WHERE     ( (SELECT PNO
                    FROM  WORKS_ON
                    WHERESSN=ESSN)
                  CONTAINS
                  (SELECT PNUMBER
                  FROM   PROJECT
                  WHERE DNUM=5) )
```

CORRELATED NESTED QUERIES (cont.)

- In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5
- The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different *for each employee tuple* because of the correlation

THE EXISTS FUNCTION

- The EXISTS operator/function is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records.
- We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12B: SELECT  FNAME, LNAME
            FROM   EMPLOYEE
            WHERE EXISTS  (SELECT *
                                     FROM  DEPENDENT
                                     WHERE SSN=ESSN AND FNAME=DEPENDENT_NAME)
```

```
_Q12: SELECT  E.FNAME, E.LNAME
        FROM    EMPLOYEE AS E
       WHERE   E.SSN IN (SELECT  ESSN
                         FROM        DEPENDENT
                          WHERE      ESSN=E.SSN AND E.FNAME=DEPENDENT_NAME)
```

THE EXISTS FUNCTION (cont.)

- 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)
```

  - In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected
  - EXISTS is necessary for the expressive power of SQL

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q13:        SELECT  DISTINCT ESSN
            FROM  WORKS_ON
            WHERE  PNO IN  (1, 2, 3)
```

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .
- Query 14: Retrieve the names of all employees who do not have supervisors.

```
Q14:          SELECT  FNAME, LNAME
              FROM    EMPLOYEE
              WHERE   SUPERSSN  IS  NULL
```

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

Joined Relations Feature in SQL2

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join
- While joining there should be some common attribute.
- Allows the user to specify different types of joins

1. regular "theta" JOIN
2. NATURAL JOIN
3. LEFT OUTER JOIN
4. RIGHT OUTER JOIN
5. FULL OUTER JOIN
6. CROSS JOIN

NATURAL JOIN

- In JOIN, only combinations of tuples satisfying the join condition appear in the result
- In the CARTESIAN PRODUCT all combinations of tuples are included in the result.
- The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples.
- Each tuple combination for which the join condition evaluates to TRUE is included in the resulting relation Q as a single combined tuple.

NATURAL JOIN

Features of natural join

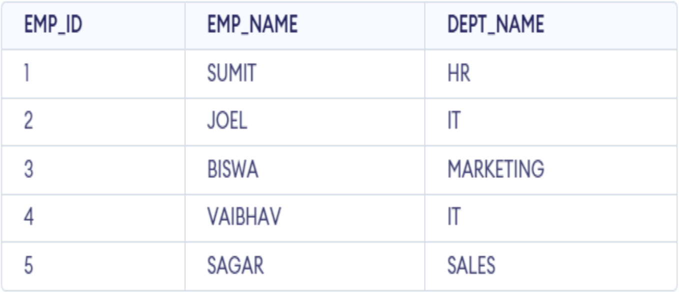
1. It will perform the Cartesian product.
2. It finds consistent tuples and deletes inconsistent tuples.
3. Then it deletes the duplicate attributes.

- If we join R1 and R2 on equal condition then it is called natural join or equi join.
- Natural join of R1 and R2 is – R1 NATURAL JOIN R2

```
SELECT  *  FROM TABLE1
NATURAL JOIN TABLE2;
```

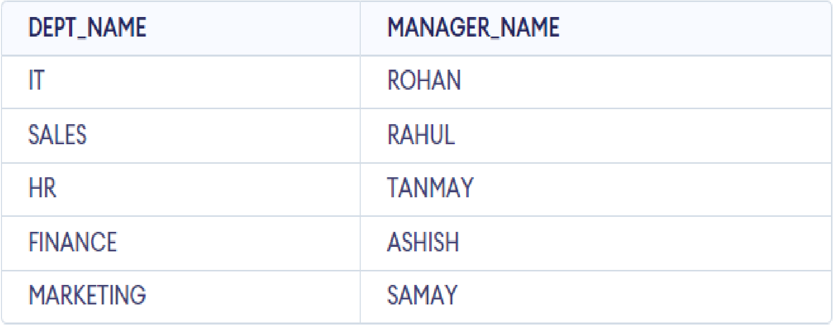
NATURAL JOIN

SELECT * FROM employee;



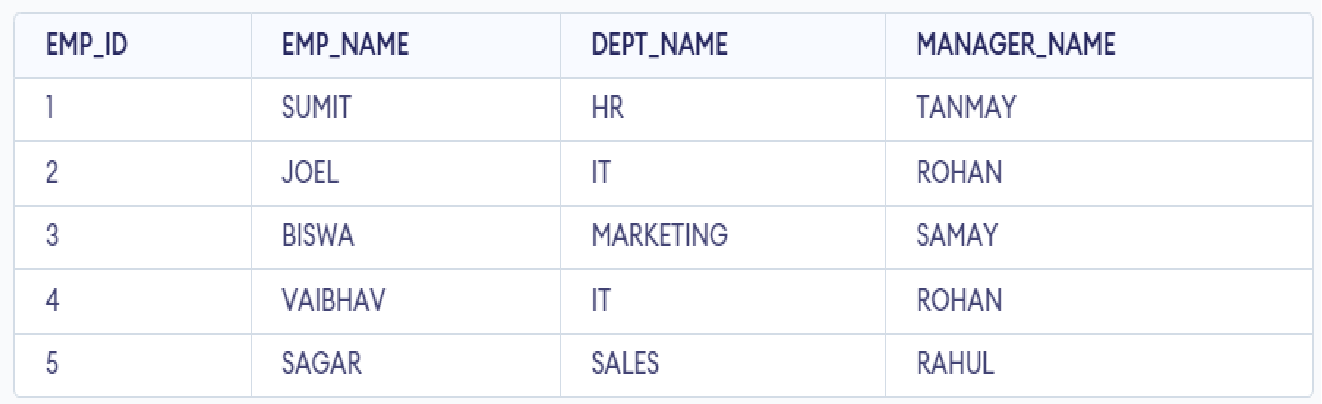
EMP_ID	EMP_NAME	DEPT_NAME
1	SUMIT	HR
2	JOEL	IT
3	BISWA	MARKETING
4	VAIBHAV	IT
5	SAGAR	SALES

SELECT * FROM department;



DEPT_NAME	MANAGER_NAME
IT	ROHAN
SALES	RAHUL
HR	TANMAY
FINANCE	ASHISH
MARKETING	SAMAY

```
SELECT *
FROM employee
NATURAL JOIN department;
```



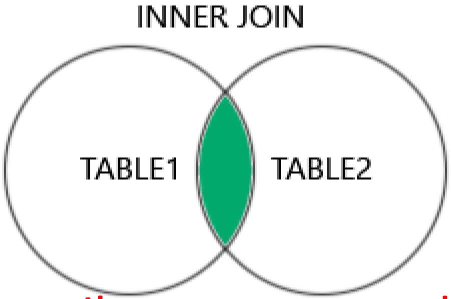
EMP_ID	EMP_NAME	DEPT_NAME	MANAGER_NAME
1	SUMIT	HR	TANMAY
2	JOEL	IT	ROHAN
3	BISWA	MARKETING	SAMAY
4	VAIBHAV	IT	ROHAN
5	SAGAR	SALES	RAHUL

INNER JOIN (equi-join)

- Inner Join in SQL is the most common type of join.
- Returns records that have matching values in both tables, i.e., combines the table based on the **common columns and selects the records that have matching values in these columns.**
- It is similar to the intersection of the sets in Mathematics. i.e. when you take the intersection of two or more sets only the common element (in all the sets) are taken together.
-

  Inner Join joins two tables on the basis of the column which is explicitly specified in the ON clause. The resulting table will contain all the attributes from both tables including the common column also.

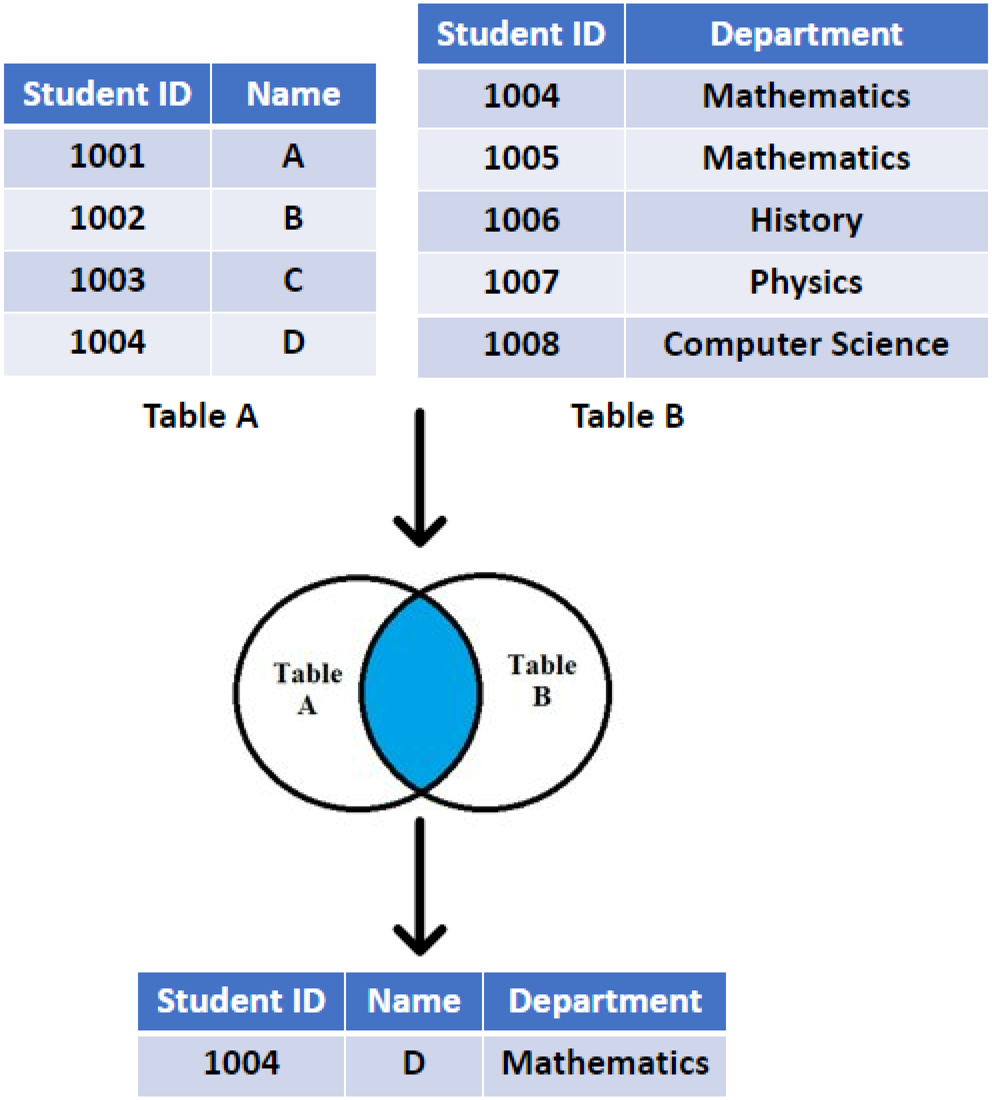
- SELECT*column_name(s)*
  FROM *table1*
  INNER JOIN *table2*
  ON *table1.column_name* = *table2.column_name*;



- Natural Join joins two tables based on the same attribute name and datatypes. The resulting table will contain all the attributes of both the table but keep only one copy of each common column.

```
SELECT Student.StudentID, Student.Name,Department.DepartmentName
FROM Student INNER JOIN Department
ON Student.DepartmentID = Department.DepartmentID
```

INNERJOIN- Returns records that have matching values in both tables, i.e., combines the table based on the common columns and selects the records that have matching values in these columns.



Student ID	Name
1001	A
1002	B
1003	C
1004	D

Table A

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science

Table B

Student ID	Name	Department
1004	D	Mathematics

```
SELECT Student.StudentID, Student.Name, Student.Email, Student.Percentage,
Department.DepartmentName
FROM Student INNER JOIN Department
ON Student.DepartmentID = Department.DepartmentID
```

Student Record Table

StudentID	Name	E-mail	Percentage(%)	DepartmentID
1001	Ajay	ajay@xyz.com	85	1
1002	Babloo	babloo@xyz.com	67	2
1003	Chhavi	chhavi@xyz.com	89	3
1004	Dheeraj	dheeraj@xyz.com	75	
1005	Evina	evina@xyz.com	91	1
1006	Krishna	krishna@xyz.com	99	5

Department Record Table

Department ID	Department Name
1	Mathematics
2	Physics
3	English

StudentID	Name	E-mail	Percentage(%)	DepartmentName
1001	Ajay	ajay@xyz.com	85	Mathematics
1002	Babloo	babloo@xyz.com	67	Physics
1003	Chhavi	chhavi@xyz.com	89	English
1005	Evina	evina@xyz.com	91	Mathematics

OUTER JOIN

- **Outer join**: It is an extension of natural join to deal with missing values of relation.
  - **Left Join:** Returns all records from the left table, and the matched records from the right table.
  - **Right Join:** Returns all records from the right table, and the matched records from the left table.
  - **Full Join:** Returns all records when there is a match in either left or right table

Left Join: Left Join in SQL is used to return all the rows from the left table but only the matching rows from the right table where the join condition is fulfilled.

- Here all the tuples of R1(left table) appear in output.
- The mismatching values of R2 are filled with NULL.
- Left outer join = natural join + mismatch / extra tuple of R1

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

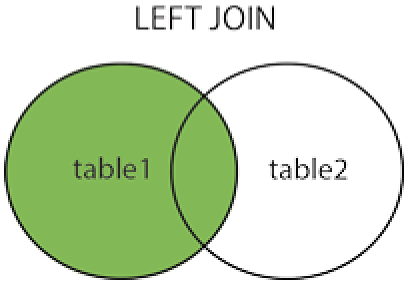


Table R1

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B
5	IT	A

Table R2

Name	Regno
Bhanu	2
Priya	4
Hari	7

Left outer join

```
SELECT RegNo, Branch, Section, Name
FROM table R1
LEFT JOIN tableR2
ON tableR1.RegNo = tableR2.RegNo;
```

RegNo	Branch	Section	Name	Regno
2	-	-	Bhanu	2
4	-	-	Priya	4
1	-	-	NULL	NULL
3	-	-	NULL	NULL
5	-	-	NULL	NULL

```
Q1:        SELECT  E.FNAME, E.LNAME, S.FNAME, S.LNAME
           FROM    EMPLOYEE E S
           WHERE E.SUPERSSN=S.SSN
```

can be written as:

```
Q2:        SELECT  E.FNAME, E.LNAME, S.FNAME, S.LNAME
           FROM    (EMPLOYEE E LEFT OUTER JOIN EMPLOYEES
                   ON  E.SUPERSSN=S.SSN)
```

Select all customers, and any orders they might have:

Example

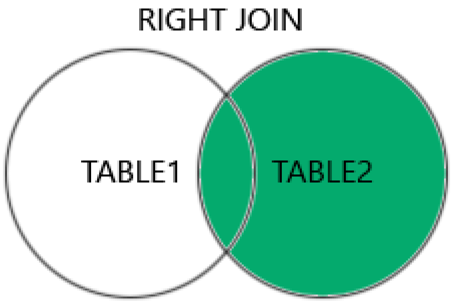
```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

The **LEFT JOIN** keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

- **Right Join**: Right Join in SQL is used to return all the rows from the right table but only the matching rows from the left table where the join condition is fulfilled.

- Here all the tuples of S(right table) appear in output. The mismatching values of S are filled with NULL.

-
  ```
  SELECT column_name(s)
  FROM table1
  RIGHT JOIN table2
  ON table1.column_name = table2.column_name;
  ```



```
SELECT RegNo, Branch, Section, Name
FROM table R1
RIGHT JOIN table R2
ON table R1.RegNo = table R2.RegNo;
```

Table R1

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B
5	IT	A

Table R2

Name	Regno
Bhanu	2
Priya	4
Hari	7

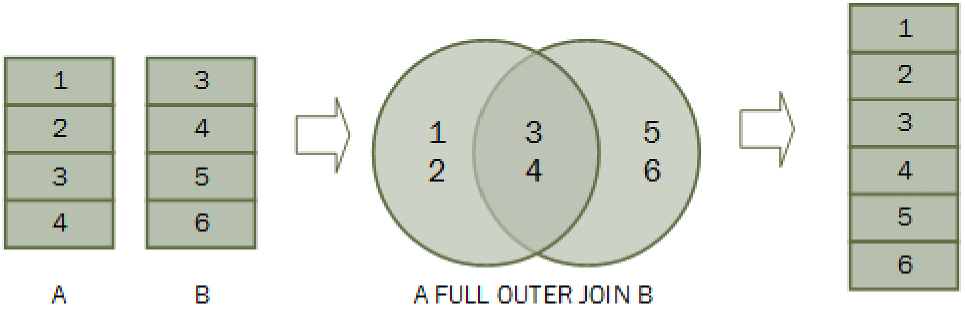
Right outer join

RegNo	Branch	Section	Name	Regno
2	-	-	Bhanu	2
4	-	-	Priya	4
NULL	NULL	NULL	Hari	7

Here all the tuples of R2(right table) appear in output. The mismatching values of R1 are filled with NULL.

Full Join: Full join returns all the records when there is a match in any of the tables. Therefore, it returns all the rows from the left-hand side table and all the rows from the right-hand side table.

Full outer join=left outer join U right outer join.



```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Table R1

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B
5	IT	A

Table R2

Name	Regno
Bhanu	2
Priya	4
Hari	7

Full outer join

```
SELECT RegNo
FROM tableR1
FULL OUTER JOIN tableR2
ON tableR1.RegNo = tableR2.RegNo
WHERE condition;
```

RegNo	Branch	Section	Name	Regno
2	-	-	Bhanu	2
4	-	-	Priya	4
1	-	-	NULL	NULL
3	-	-	NULL	NULL
5	-	-	NULL	NULL
NULL	NULL	NULL	Hari	7

THETA JOIN(non equi-join)

- A theta join is a join that links tables based on a relationship other than equality between two columns.
- A theta join could use any operator other than the "equal" operator.
- A general join condition is of the form

<condition> **AND** <condition> **AND**...**AND** <condition>

where each <condition> is of the form $Ai\ \theta\ Bj$, Ai is an attribute of R, Bj is an attribute of S, Ai and Bj have the same domain, and θ (theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$.

- A JOIN operation with such a general join condition is called a **THETA JOIN**.

Table R1

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B
5	IT	A

Table R2

Name	RegNo
Bhanu	2
Priya	4

```
SELECT RegNo
FROM tableR1, tableR2
WHERE (tableR1.RegNo BETWEEN tab
leR2.RegNo AND table R1. branch,
table R1.section AND
tableR1.name;
```

- R1 thetajoin R2 with condition R1.regno > R2.regno
- In the join operation, we select those rows from the cartesian product where R1.regno>R2.regno.
- Join operation = select operation + cartesian product operation

RegNo	Branch	Section	Name	Regno
3	CIVIL	A	Bhanu	2
4	IT	B	Bhanu	2
5	IT	A	Bhanu	2
5	IT	B	Priya	4

In theta join we join relations R1 and R2 other than the equal to condition

DIVISION Operation

- The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications.
- DIVISION operation is applied to two relations $R(Z) \div S(X)$,
- Where the attributes of R are a subset of the attributes of S; that is, $X \subseteq Z$. Let Y be the set of attributes of R that are not attributes of S; that is, $Y = Z – X$ (and hence $Z = X \cup Y$).
- For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S.

Joined Relations Feature in SQL2 (cont.)

- Examples:

```
Q8:
Q1:   SELECT   FNAME, LNAME, ADDRESS
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DNAME='Research' AND DNUMBER=DNO
```

Joined Relations Feature in SQL2 (cont.)

- could be written as:

```
Q1:  SELECT  FNAME, LNAME, ADDRESS
     FROM    (EMPLOYEE JOIN DEPARTMENT
              ON DNUMBER=DNO)
     WHERE   DNAME='Research’
```

or as:

```
Q1:  SELECT  FNAME, LNAME, ADDRESS
     FROM    (EMPLOYEE NATURAL JOIN DEPARTMENT
              AS DEPT(DNAME, DNO, MSSN, MSDATE)
     WHERE   DNAME='Research’
```

Joined Relations Feature in SQL2 (cont.)

- Another Example;

–Q2 could be written as follows; this illustrates multiple joins in the joined tables

```
Q2:SELECT  PNUMBER, DNUM, LNAME, BDATE, ADDRESS
     FROM (PROJECT JOIN  DEPARTMENT ON DNUM=DNUMBER)
              JOIN  (EMPLOYEE ON  MGRSSN=SSN) )
    WHERE  PLOCATION='Stafford’
```

AGGREGATE FUNCTIONS

- An aggregate function in SQL performs a **calculation on multiple values and returns a single value.**
- SQL provides many aggregate functions that include AVG, COUNT, SUM, MIN, MAX, etc.
- An aggregate function ignores NULL values when it performs the calculation, except for the count function.

- Query 15a: Find the salary, among all employees.
- **Q15: SELECT ***

  FROM EMPLOYEE

- Query 15b: Find the maximum salary, among all employees.
- **Q15: SELECT MAX(SALARY)**
  FROM EMPLOYEE

AGGREGATE FUNCTIONS

- Query 15c: Find the maximum salary, the minimum salary, and the average salary among all employees.
- **Q15: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)**
  FROM EMPLOYEE

- Some SQL implementations *may not allow more than one function* in the SELECT-clause
- Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

- **Q16: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)**
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNO=DNUMBER AND DNAME='Research'

AGGREGATE FUNCTIONS (cont.)

- The COUNT() aggregate function returns the total number of rows that match the specified criteria.
- i.e., **get the number of rows** for a particular group in the table.
- Here is the basic syntax:

```
SELECT COUNT(column_name)
FROM table_name;
```

- COUNT(column_name) will not include NULL values as part of the count.

```
SELECT COUNT(ProductID)
FROM Products;
```

AGGREGATE FUNCTIONS (cont.)

- The COUNT(*) function will **return the total number of items** in that group including **NULL values**.
- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17:   SELECT  COUNT (*)
       FROM    EMPLOYEE

Q18:   SELECT  COUNT (*)
       FROM    EMPLOYEE, DEPARTMENT
       WHERE   DNO=DNUMBER AND DNAME='Research'
```

ORDER BY

- The ORDER BY clause is **used to sort the tuples** in a query result based on the values of some attribute(s)
- Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q28: SELECT      DNAME, LNAME, FNAME, PNAME
     FROM        DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
     WHERE       DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
     ORDER BY    DNAME, LNAME
```

- The default order is in ascending order of values
- We can specify the keyword DESC if we want a **descending order**; the keyword ASC can be used to explicitly specify **ascending order**, even though it is the default.

GROUPING

***GROUP BY* clause is used in conjunction with the *SELECT* statement and *aggregate function* to group rows together by common values**

- In many cases, we want to apply the **aggregate functions** *to* **subgroups of tuples in a relation**
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the **SELECT-clause***

GROUPING (cont.)

- Query 20a: For each department, retrieve the department number.

  ```
  Q20: SELECT   DNO
       FROM     EMPLOYEE
       GROUP BY DNO
  ```

- Query 20b: For each department, retrieve the department number, the number of employees in the department, and their average salary.

  ```
  Q20: SELECT   DNO, COUNT (*), AVG (SALARY)
       FROM     EMPLOYEE
       GROUP BY DNO
  ```

  In Q20 the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO

  - The COUNT and AVG functions are applied to each such group of tuples separately
  - The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
  - A join condition can be used in conjunction with grouping

GROUPING (cont.)

- Query 21a: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21: SELECT     PNUMBER, PNAME, COUNT (*)
     FROM       PROJECT, WORKS_ON
     WHERE      PNUMBER=PNO
     GROUP BY PNUMBER, PNAME
```

In this case, the grouping and functions are applied *after* the joining of the two relations

- Query 21b: For each project, retrieve the project number, project name, and the number of employees who work on that project. Sort by high to low

```
Q21: SELECT     PNUMBER, PNAME, COUNT (*)
     FROM        PROJECT, WORKS_ON
     WHERE      PNUMBER=PNO
     GROUP BY PNUMBER, PNAME
     ORDER BY COUNT(PNUMBER)  DESC;
```

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The HAVING-clause is used for specifying a **selection condition on groups** (rather than on **individual tuples**)
- **The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.**

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

THE HAVING-CLAUSE

- Query 22: For each project on which more than two employees work , retrieve the project number, project name, and the number of employees who work on that project.

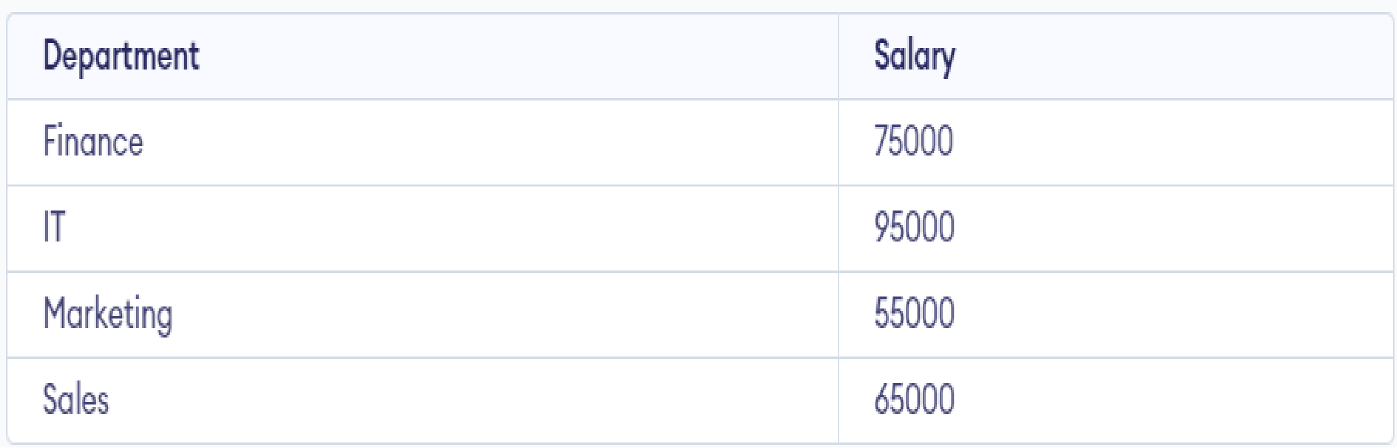
-
```
Q22: SELECT       PNUMBER, PNAME, COUNT (*)
     FROM         PROJECT, WORKS_ON
     WHERE        PNUMBER=PNO
     GROUP BY     PNUMBER, PNAME
     HAVING       COUNT (*) > 2
```

- The main difference between WHERE and HAVING clause is that the **WHERE clause allows you to filter data from specific rows** (individual rows) from a table **based on certain conditions**.
- In contrast, the **HAVING clause allows you to filter data from a group of rows** in a query **based on conditions involving aggregate values**.

THE HAVING-CLAUSE

Display the departments where the sum of salaries is 50,000 or more.

```
SELECT Department, SUM(Salary) as Salary
FROM employee
GROUP BY department
HAVING SUM(Salary) >= 50000;
```



Department	Salary
Finance	75000
IT	95000
Marketing	55000
Sales	65000

HAVING	WHERE
In the HAVING clause it will check the condition in group of a row.	In the WHERE condition it will check or execute at each row individual.
HAVING clause can only be used with aggregate function.	The WHERE Clause cannot be used with aggregate function like Having
Priority Wise HAVING Clause is executed after Group By.	Priority Wise WHERE is executed before Group By.

SUBSTRING COMPARISON

- The **LIKE** comparison operator is used to compare partial strings
- Two reserved characters are used: '%' (or '*' in some implementations) replaces an arbitrary number of characters, and '_' replaces a single arbitrary character.

- Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.

```
Q25:SELECT   FNAME, LNAME
    FROM     EMPLOYEE
    WHERE    ADDRESS LIKE  '%Houston,TX%’
```

SUBSTRING COMPARISON (cont.)

- Query 26: Retrieve all employees who were born during the 1950s. Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '_______5_', with each underscore as a place holder for a single arbitrary character.
-

```
Q26: SELECT  FNAME, LNAME
     FROM    EMPLOYEE
     WHERE   BDATE LIKE  '_______5_’
```

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible; hence, in SQL, character string attribute values are not atomic

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

```
SELECT      <attribute list>
FROM        <table list>
[WHERE      <condition>]
[GROUP BY <grouping attribute(s)>]
[HAVING     <group condition>]
[ORDER BY <attribute list>]
```

Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause