MODULE III

SQL DATA DEFINITION

SQL uses the terms table, row, and column for the formal relational model terms relation, tuple, and attribute, respectively.

The name SQL is derived from Structured Query Language. Originally, SQL was called SEQUEL (for Structured English QUEry Language) and was designed and implemented at IBM Research.

SQL is now the standard language for commercial relational DBMSs.

SQL is a comprehensive database language: It has statements for data definition, query, and update. Hence, it is both a DOL and a DML. In addition, it has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls.

ATTRIBUTE DATA TYPES AND DOMAINS IN SQL

(a)Numeric data types

- **integer** numbers of various sizes (INTEGER or INT, and SMALLINT)

-**floating-point** (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION).

- Formatted numbers can be declared by using **DECIMAL(i,j)**or DEC(i,j) or NUMERIC(i,j)- where **i, the precision**, is the total number of decimal digits and **j, the scale**, is the number of digits after the decimal point.

(b)Character-string data type

-either **fixed length--**CHAR(n) or CHARACTER(n), where n is the number of characters

- **varying** length-VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters.

(c)Bit-string data types

BIT(n)-or varying length-BIT VARYING(n), Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings; for example, B'10101’

(d)Boolean data type -TRUE ,FALSE,Unknown

In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a boolean data type is UNKNOWN.

(e)**Date and Time datatypes**

The DATE data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD. The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS.

(f)Timestamp data type (TIMESTAMP)

A timestamp data type (TIMESTAMP) includes both the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds.

Example: TIMESTAMP '2002-09-27 09:12:4 7648302'.

SCHEMA AND CATALOG CONCEPTS IN SQL

An SQL schema is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for each element in the schema.

Schema elements include tables, constraints, views, domains, and other constructs (such as authorization grants) that describe the schema. For example, the following statement creates a schema called COMPANY, owned by the user with authorization identifier JSMITH:

CREATE SCHEMA COMPANY AUTHORIZATION JSMITH;

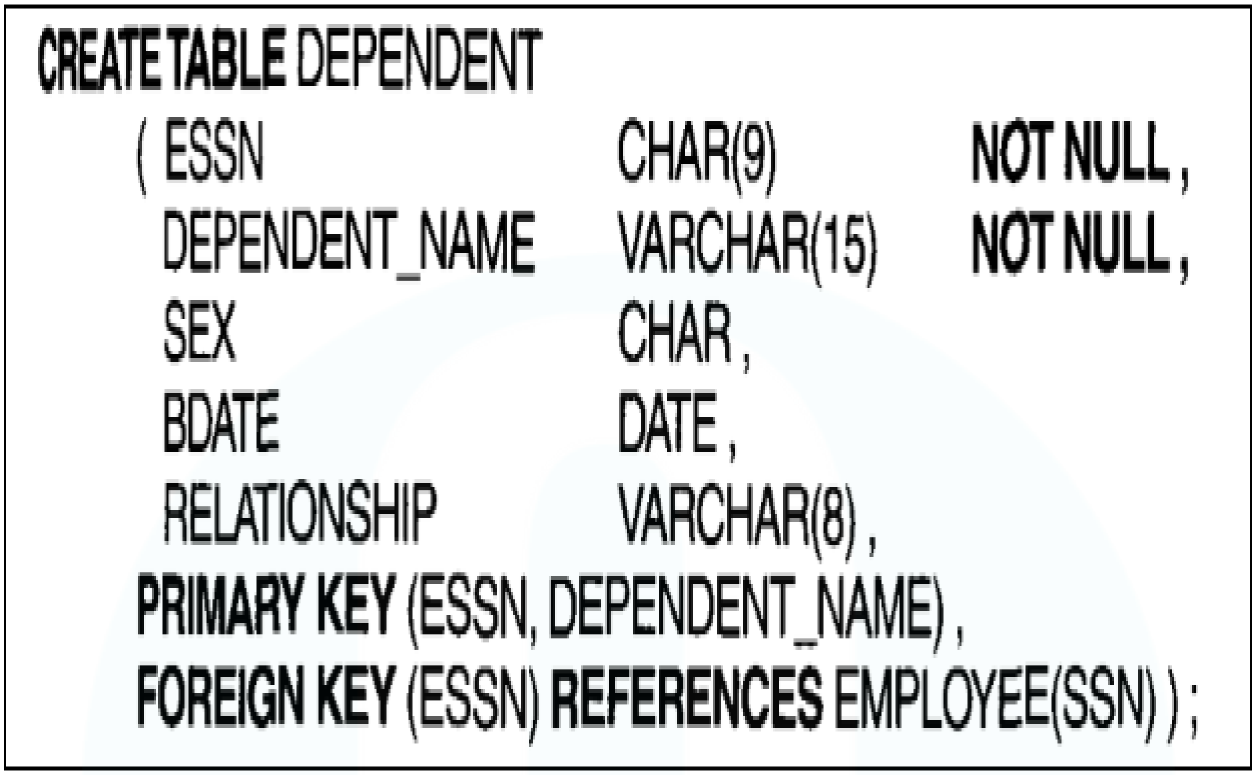
In general, not all users are authorized to create schemas and schema elements. The privilege to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.

CREATE TABLE COMMAND

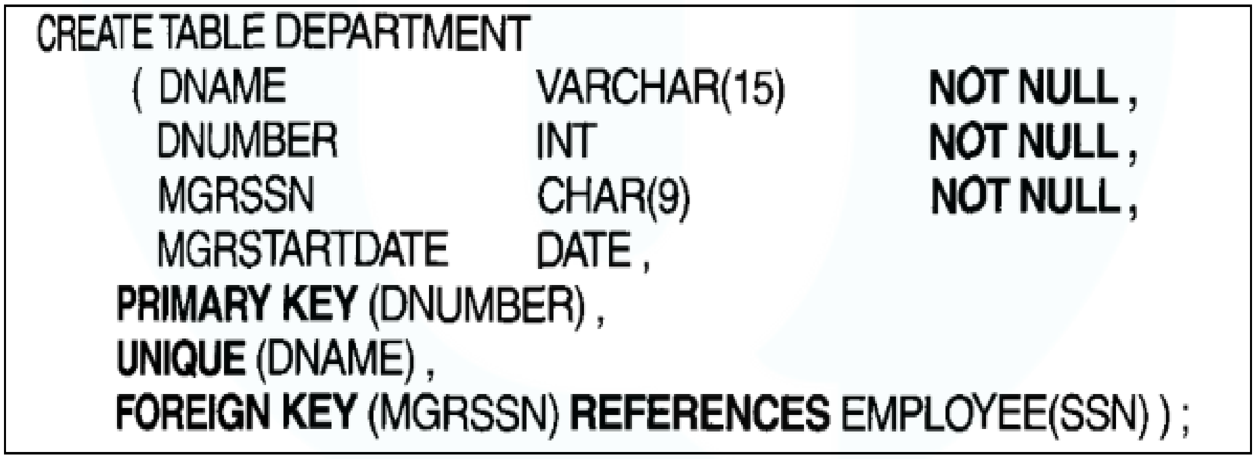
The syntax is given below:

```
CREATE TABLE <table name> (<column name> <column type> [<attribute constraint>]
                          {, <column name> <column type> [<attribute constraint>] }
                          [<table constraint> {,<table constraint>}])
```

```
CREATE TABLE EMPLOYEE
        ( FNAME          VARCHAR(15)        NOT NULL ,
          MINIT          CHAR ,
          LNAME          VARCHAR(15)        NOT NULL ,
          SSN            CHAR(9)            NOT NULL ,
          BDATE          DATE,
          ADDRESS        VARCHAR(30) ,
          SEX            CHAR ,
          SALARY         DECIMAL(10,2) ,
          SUPERSSN       CHAR(9) ,
          DNO            INT                NOT NULL ,
  PRIMARY KEY (SSN) ,
  FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN) ,
  FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER) ) ;
```



```
CREATE TABLE DEPENDENT
    ( ESSN              CHAR(9)        NOT NULL ,
      DEPENDENT_NAME    VARCHAR(15)    NOT NULL ,
      SEX               CHAR ,
      BDATE             DATE ,
      RELATIONSHIP      VARCHAR(8) ,
    PRIMARY KEY (ESSN, DEPENDENT_NAME) ,
    FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) ) ;
```



```
CREATE TABLE DEPARTMENT
    ( DNAME             VARCHAR(15)    NOT NULL ,
      DNUMBER           INT            NOT NULL ,
      MGRSSN            CHAR(9)        NOT NULL ,
      MGRSTARTDATE      DATE ,
    PRIMARY KEY (DNUMBER) ,
    UNIQUE (DNAME) ,
    FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN) ) ;
```

SPECIFYING BASIC CONSTRAINTS IN SQL

1.Specifying Attribute Constraints and Attribute Defaults

Because SQL allows NULLs as attribute values, a constraint NOT NULL may be specified if NULL is not permitted for a particular attribute. This is always implicitly specified for the attributes that are part of the primary key of each relation, but it can be specified for any other attributes whose values are required not to be NULL.

It is also possible to define a default value for an attribute by appending the clause DEFAULT <value> to an attribute definition.

If no default clause is specified, the default, default value is NULL for attributes that do not have the NOT NULL constraint.

Another type of constraint can restrict attribute or domain values using the CHECK clause following an attribute or domain definition.6 For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then, we can change the attribute declaration of DNUMBER in the DEPARTMENT table

DNUMBER INT NOT NULL CHECK (DNUMBER > 0 AND DNUMBER < 21);

2.Specifying Key and Referential Integrity Constraints

The PRIMARY KEY clause specifies one or more attributes that make up the primary key of a relation. Ifa primary key has a single attribute, the clause can follow the attribute directly.

For example, the primary key of DEPARTMENT can be specified as follows:

DNUMBER INT PRIMARY KEY

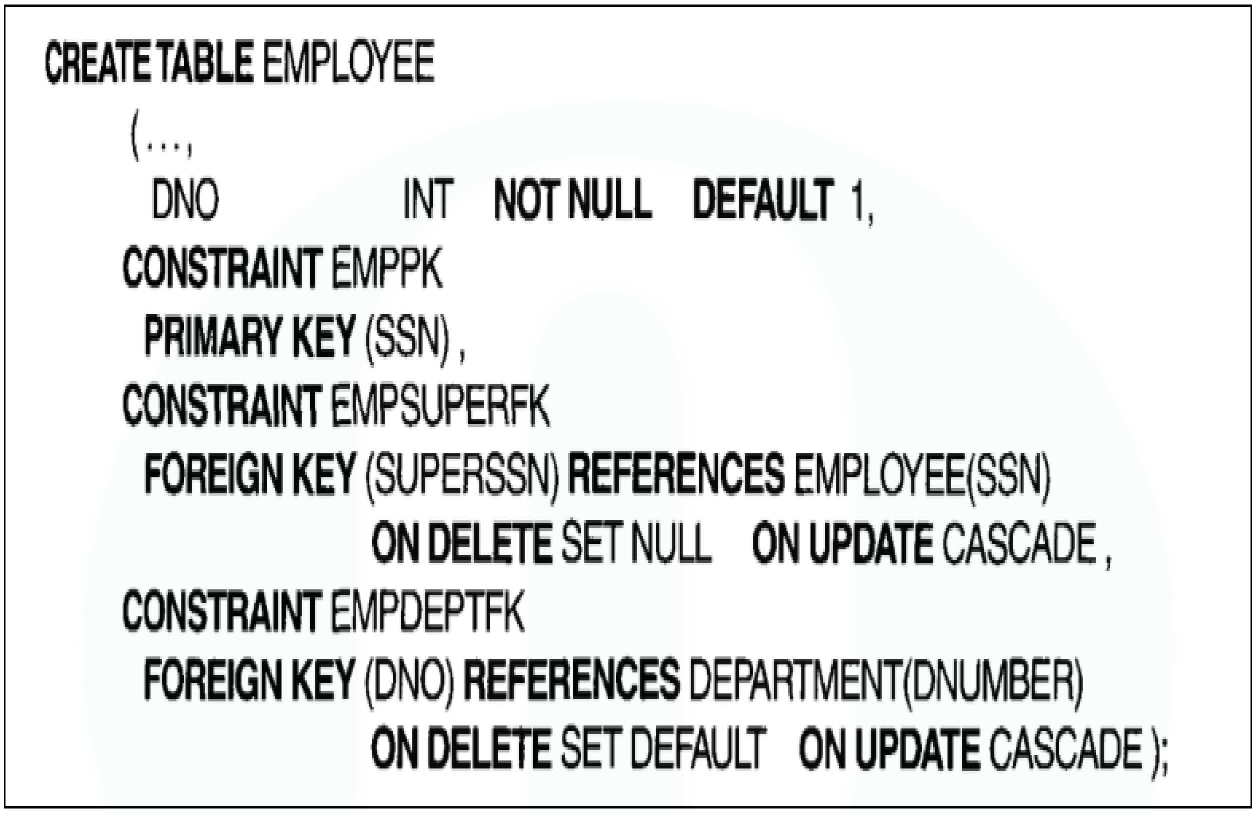
The UNIQUE clause specifies alternate (secondary) keys

Referential integrity is specified via the FOREIGN KEY clause . a referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is modified.

The default action that SQL takes for an integrity violation is to reject the update operation that will cause a violation. However, the schema designer can specify an alternative action to be taken if a referential integrity constraint is violated, by attaching a referential triggered action clause to any foreign key constraint. The options include SET NULL, CASCADE, and SET DEFAULT. An option must be qualified with either ON DELETE or ON UPDATE.

In Figure below, the database designer chooses SET NULL ON DELETE and CASCADE ON UPDATE for the foreign key SUPERSSN of EMPLOYEE. This means that if the tuple for a supervising employee is deleted, the value of SUPERSSN is automatically set to NULL for all employee tuples that were referencing the deleted employee tuple. On the other hand, if the SSN value for a supervising employee is updated (say, because it was entered incorrectly), the new value is cascaded to SUPERSSN for all employee tuples referencing the updated employee tuple.

The action for CASCADE ON DELETE is to delete all the referencing tuples, whereas the action for CASCADE ON UPDATE is to change the value of the foreign key to the updated (new) primary key value for all referencing tuples.



```
CREATE TABLE EMPLOYEE
      (...,
       DNO          INT   NOT NULL   DEFAULT 1,
      CONSTRAINT EMPPK
       PRIMARY KEY (SSN) ,
      CONSTRAINT EMPSUPERFK
       FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN)
                    ON DELETE SET NULL   ON UPDATE CASCADE ,
      CONSTRAINT EMPDEPTFK
       FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER)
                    ON DELETE SET DEFAULT   ON UPDATE CASCADE );
```

3.Giving Names to Constraints

How a constraint may be given a constraint name, following the keyword **CONSTRAINT**.

A constraint name is used to identify a particular constraint in case the constraint must be dropped later and replaced with another constraint

4.Specifying Constraints on Tuples Using CHECK

In addition to key and referential integrity constraints, which are specified by special keywords, other table constraints can be specified through additional CHECK clauses at the end of a CREATE TABLE statement.

Then we could add the following CHECK clause at the end of the CREATE TABLE statement for the DEPARTMENT table to make sure that a manager's start date is later than the department creation date:

CHECK (DEPT_CREATE_DATE < MGRSTARTDATE);

THE SELECT-FROM-WHERE STRUCTURE OF BASIC SQL QUERIES

The syntax is given below:

```
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.

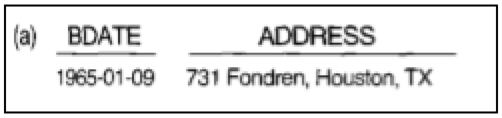
Syntax in detail:

```
SELECT <ATTRIBUTE AND FUNCTION LIST>
FROM <TABLE LIST>
[WHERE <CONDITION>]
[GROUP BY <GROUPING ATTRIBUTE(S)>]
[HAVING <GROUP CONDITION>]
[ORDER BY <ATTRIBUTE LIST>];
```

Query 1

Retrieve the birthdate and address of the ernploveeis) whose name is 'John B. Smith'.

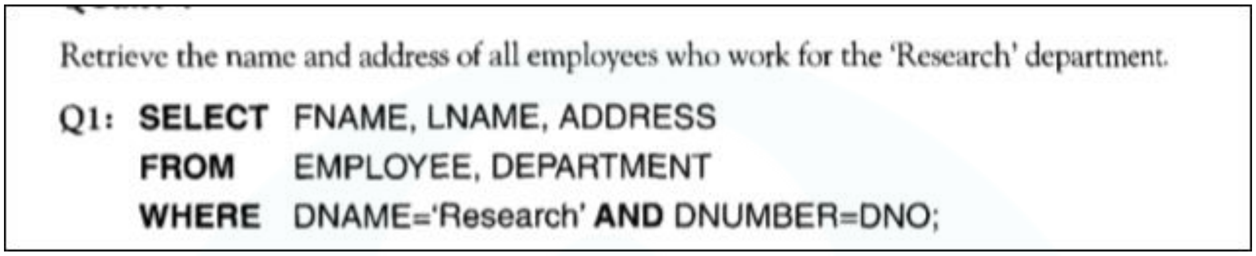
```
SELECT   BDATE, ADDRESS
FROM     EMPLOYEE
WHERE    FNAME='John' AND MINIT='B' AND LNAME='Smith';
```



(a)

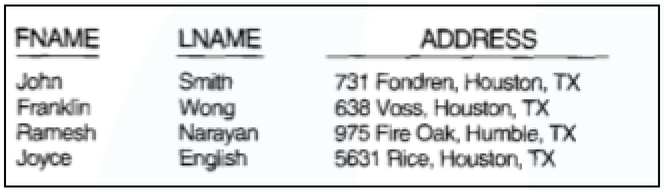
BDATE	ADDRESS
1965-01-09	731 Fondren, Houston, TX

Query 2



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;
```



FNAME	LNAME	ADDRESS
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

<u>AMBIGUOUS ATTRIBUTE NAMES, ALIASING</u>

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 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(.).

For example ,let 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,

Then the Query 2 above can be rewritten as

SELECT fname, employee.name, address

FROM employee,department

WHERE department.name='research' and department.dnumber=employee.dnumber;

To rename the relation attributes within the query in SQL by giving them aliases. For example, if we write

EMPLOYEE AS E(FN, MI, LN, SSN, SD, ADDR, SEX, SAL, SSSN, DNO)

```
SELECT   E.FNAME, E.NAME, E.ADDRESS
FROM     EMPLOYEE E, DEPARTMENT D
WHERE    D.NAME='Research' AND D.DNUMBER=E.DNUMBER;
```

Here,aliasing is done with employee and department table names.Rather than using “employee “ and “department”everywhere we can make use of aliases “E ” and ”D”respectively.

UNSPECIFIED WHERE CLAUSE

Select all EMPLOYEE SSNS (Q9), and all combinations of EMPLOYEE SSN and DEPARTMENT DNAME (Q10) in the database.

```
Q9:   SELECT  SSN
      FROM    EMPLOYEE;
Q10:  SELECT  SSN, DNAME
      FROM    EMPLOYEE, DEPARTMENT;
```

USE OF THE ASTERISK (*)

```
QIC:    SELECT  *
        FROM    EMPLOYEE
        WHERE   DNO=5;
QID:    SELECT  *
        FROM    EMPLOYEE, DEPARTMENT
        WHERE   DNAME='Research' AND DNO=DNUMBER;
Q10A:   SELECT  *
        FROM    EMPLOYEE, DEPARTMENT;
```

Query QIC retrieves all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5 query QID 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.

Query 11 below retrieves the salary of every employee; if several employees have the same salary, that salary value will appear as many times in the result of the query,. If we are interested only in distinct salary values, we want each value to appear only once, regardless of how many employees earn that salary. By using the keyword DISTINCT as in QIIA, we can accomplish this.

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                    (b)  SALARY
     ------                         ------
     30000                          30000
     40000                          40000
     25000                          25000
     43000                          43000
     38000                          38000
     25000                          55000
     25000
     55000
```

(a)output of Q11

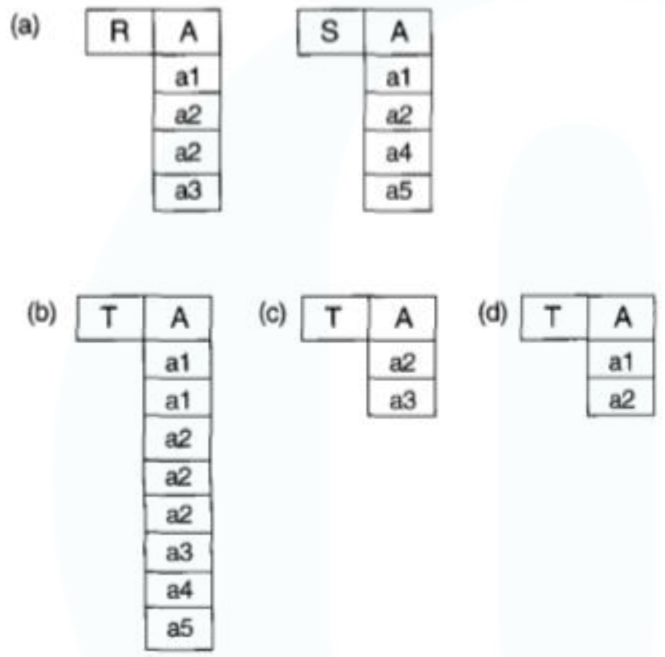
(b)output of Q11A

SET OPERATIONS

SQL has directly incorporated some of the set operations 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. Because these set operations apply only to union-compatible relations, we must make sure that the two relations on which we apply theoperation have the same attributes and that the attributes appear in the same order in both relations.

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.

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



(a)

R	A
	a1
	a2
	a2
	a3

S	A
	a1
	a2
	a4
	a5

(b)

T	A
	a1
	a1
	a2
	a2
	a2
	a3
	a4
	a5

(c)

T	A
	a2
	a3

(d)

T	A
	a1
	a2

FIGURE 8.5 The results of SQL multiset operations. (a) Two tables, R(A) and S(A). (b) R(A) UNION ALL S(A). (c) R(A) EXCEPT ALL S(A). (d) R(A) INTERSECT ALL S(A).

SUBSTRING PATTERN MATCHING AND ARITHMETIC OPERATORS

Partial strings are specified using two reserved characters:

(a)% replaces an arbitrary number of zero or more characters,

(b)the underscore (_)replaces a single character.

For example, consider the following query.

Retrieve all employees whose address is in Houston, Texas.

```
Q12:    SELECT   FNAME, LNAME
        FROM     EMPLOYEE
        WHERE    ADDRESS LIKE '%Houston,TX%';
```

Find all employees who were born during the 1950s.

```
Q12A:   SELECT   FNAME, LNAME
        FROM     EMPLOYEE
        WHERE    BDATE LIKE '_ _ 5 _ _ _ _ _ _ _';
```

To retrieve all employees who were born during the 1950s, we can use Query 12A. 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.

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.

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

```
SELECT   *
FROM     EMPLOYEE
WHERE    (SALARY BETWEEN 30000 AND 40000) AND DNO = 5;
```

<u>ORDERING OF QUERY RESULTS</u>

SQL allows the user to order the tuples in the result of a query by the values of one or more attributes, using the ORDER BY clause.

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

```
Q15:  SELECT     DNAME, LNAME, FNAME, PNAME
      FROM       DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
      WHERE      DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
      ORDER BY   DNAME, LNAME, FNAME;
```

The default order is in ascending order of values. We can specify the keyword DESCif we want to see the result in a descending order of values. The keyword ASC can be usedto specify ascending order explicitly. For example, if we want descending order on DNAME and ascending order on LNAME, FNAME, the ORDER BY clause of Q15 can be written as

ORDER BY DNAME DESC, LNAME ASC, FNAME ASC

COMPARISONS INVOLVING NULL

SQL allows queries that check whether an attribute value is NULL. Rather than using =or< >(not equal) to compare an attribute value to NULL, SQL uses IS or IS NOT.

QUERY 18

Retrieve the names of all employees who do not have supervisors.

```
Q18: SELECT  FNAME, LNAME
     FROM    EMPLOYEE
     WHERE   SUPERSSN IS NULL;
```

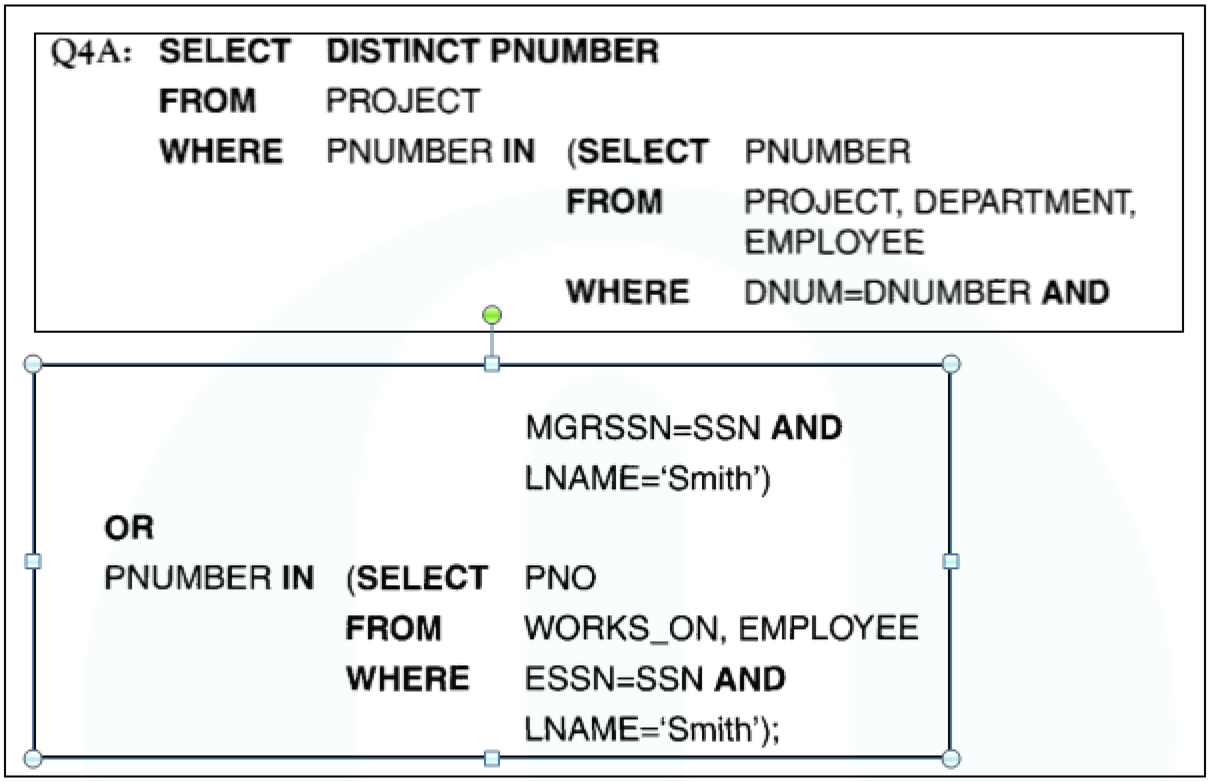
NESTED QUERIES

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.

```
(SELECT DISTINCT PNUMBER
FROM    PROJECT, DEPARTMENT, EMPLOYEE

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

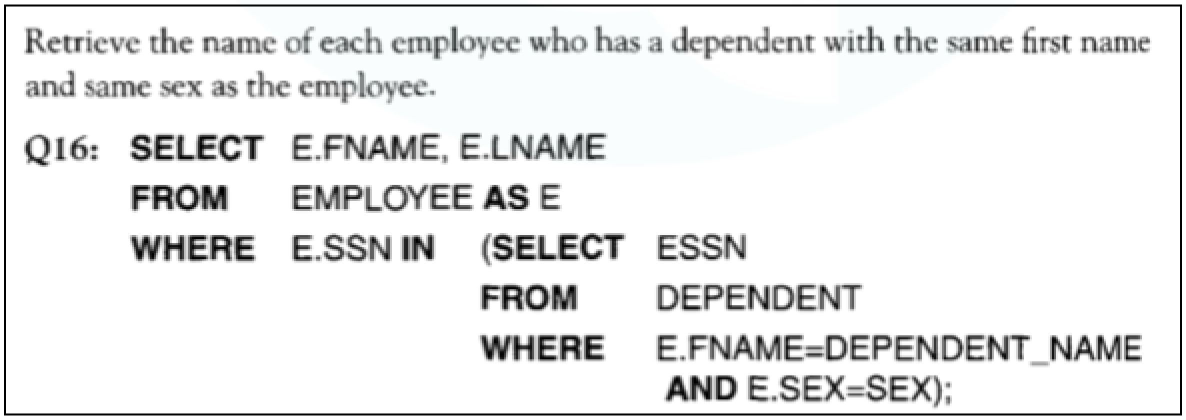
This can be rewritten with nested query as



```
Q4A: SELECT   DISTINCT PNUMBER
     FROM     PROJECT
     WHERE    PNUMBER IN  (SELECT  PNUMBER
                           FROM    PROJECT, DEPARTMENT,
                                   EMPLOYEE
                           WHERE   DNUM=DNUMBER AND
                                   MGRSSN=SSN AND
                                   LNAME='Smith')
     OR
     PNUMBER IN  (SELECT  PNO
                  FROM    WORKS_ON, EMPLOYEE
                  WHERE   ESSN=SSN AND
                          LNAME='Smith');
```

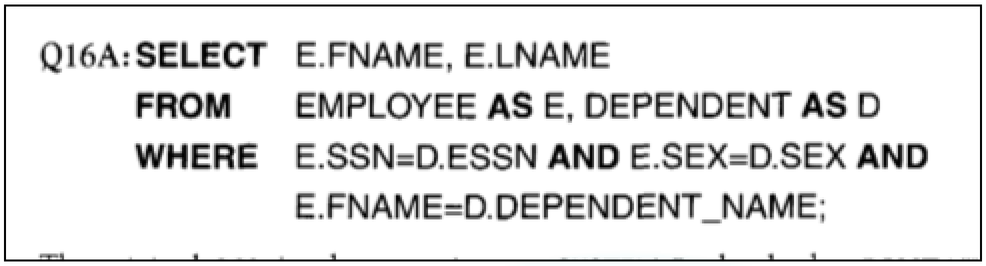
It introduces the comparison operator IN, which compares a value v with a set (or multiset) of values V and evaluates to TRUE if v is one of the elements in V.

The first nested query selects the project numbers of projects that have a 'Smith' involved as manager, while the second selects the project numbers of projects that have a 'Smith' involved as worker. In the outer query, we use the OR logical connective to retrieve aPROJECT tuple if the PNUMBER value of that tuple is in the result of either nested query.



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

```
Q16: SELECT  E.FNAME, E.LNAME
     FROM    EMPLOYEE AS E
     WHERE   E.SSN IN  (SELECT  ESSN
                        FROM    DEPENDENT
                        WHERE   E.FNAME=DEPENDENT_NAME
                                AND E.SEX=SEX);
```



```
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;
```

Q16 can be written as Q16 A without nested query.

In the nested query of Q16, we must qualify E. SEX because it refers to the SEXattribute of EMPLOYEE from the outer query, and DEPENDENT also has an attribute called SEX. All unqualified references to SEX in the nested query refer to SEXof DEPENDENT. However, we do not have to qualify FNAME and SSN because the DEPENDENT relation does not have attributes called FNAME and SSN,so there is no ambiguity.

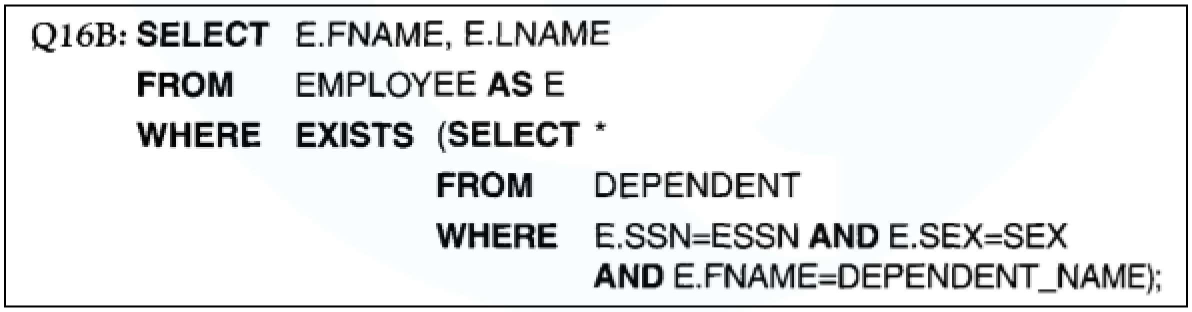
CORRELATED NESTED QUERIES

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.

We can understand a correlated query better by considering that the nested query is evaluated once for each tuple (or combination of tuples) in the outer query.

THE EXISTS AND NOT EXISTS FUNCTIONS IN SQL

Q16 can be written with EXISTS function



```
Q16B: SELECT   E.FNAME, E.LNAME
      FROM     EMPLOYEE AS E
      WHERE    EXISTS  (SELECT  *
                        FROM    DEPENDENT
                        WHERE   E.SSN=ESSN AND E.SEX=SEX
                                AND E.FNAME=DEPENDENT_NAME);
```

Retrieve the names of employees who have no dependents.

```
Q6:  SELECT  FNAME, LNAME
     FROM    EMPLOYEE
     WHERE   NOT EXISTS (SELECT  *
                         FROM    DEPENDENT
                         WHERE   SSN=ESSN);
```

EXISTS and NOTEXISTS are usually used in conjunction with a correlated nested query. In general, EXISTS(Q) returns TRUE if there is at least onetuple in the result of the nested query Q, and it returns FALSE otherwise. On the other hand, NOTEXISTS(Q) returns TRUE if there are no tuples in the result of nested query Q, and it returns FALSE otherwise.

EXPLICIT SETS

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);
```

JOINED TABLES IN SQL

Query 1 can be rewritten as

```
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 Rand S, no join condition is specified.

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

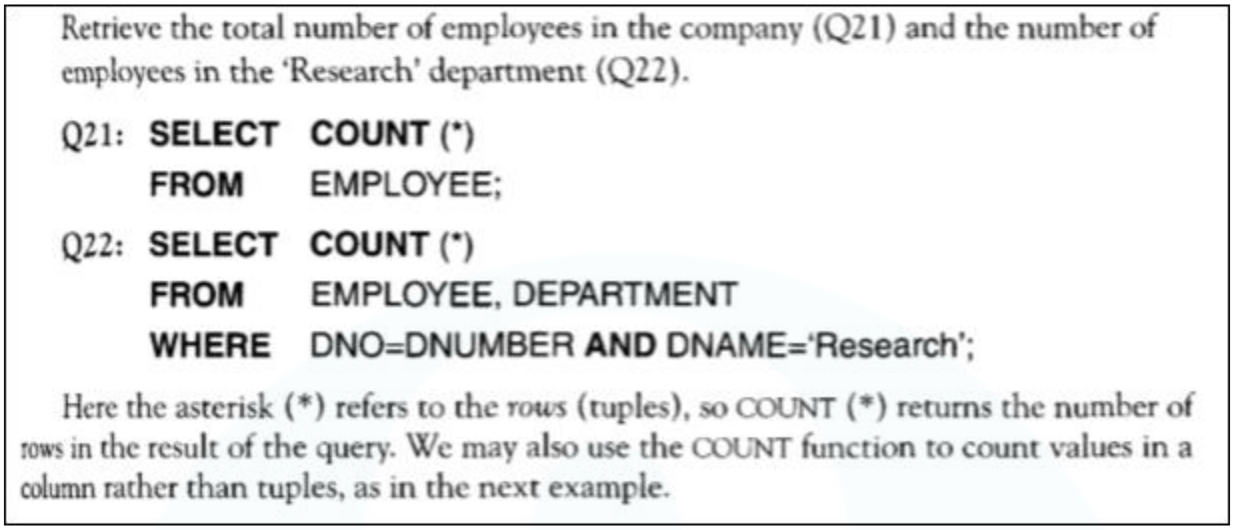
A number of built-in functions exist: COUNT, SUM, MAX, MIN, and AVG. The COUNT function returns the number of tuples or values as specified in a query. The functions SUM, MAX, MIN, and AVG are applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values. These functions can be used in the SELECT clause or in a HAVING clause.

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;
```

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';
```



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.

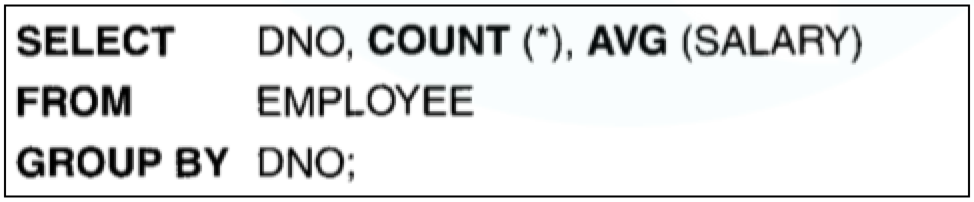
The GROUP BY and HAVING Clauses

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.

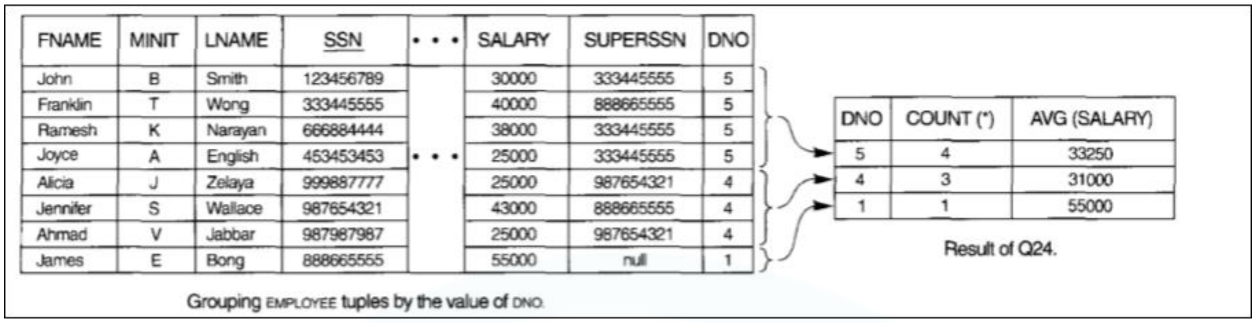
The GROUP BY clause specifies the grouping attributes, which should also appear in the SELECT clause, so that the value resulting from applying each aggregate function toa group of tuples appears along with the value of the grouping attributes

Query

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



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



FNAME	MINIT	LNAME	SSN	• • •	SALARY	SUPERSSN	DNO
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	• • •	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	null	1

Grouping EMPLOYEE tuples by the value of DNO.

DNO	COUNT (*)	AVG (SALARY)
5	4	33250
4	3	31000
1	1	55000

Result of Q24.

the EMPLOYEE tuples are partitioned 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.

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

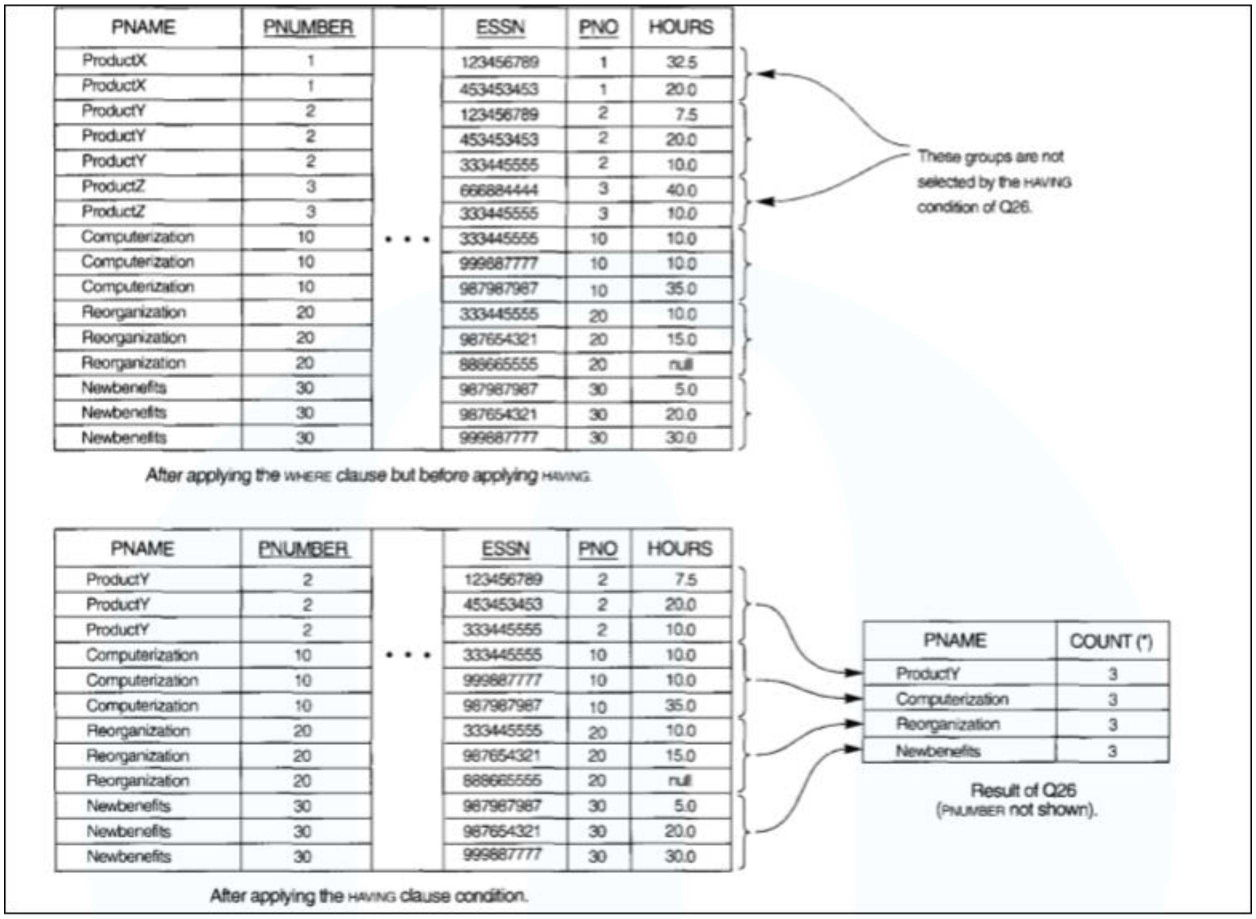
```
Q25: SELECT    PNUMBER, PNAME, COUNT (*)
     FROM      PROJECT, WORKS_ON
     WHERE     PNUMBER=PNO
     GROUP BY  PNUMBER, PNAME;
```

This query shows how we can use a join condition(Pnumber=Pno) in conjunction with GROUP BY.

HAVING provides a condition on the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result ofthequery.

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;
```



PNAME	PNUMBER		ESSN	PNO	HOURS
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	. . .	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	null
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING.

PNAME	PNUMBER		ESSN	PNO	HOURS
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10	. . .	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	null
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the HAVING clause condition.

PNAME	COUNT (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26 (PNUMBER not shown).

Notice that, while **selection conditions in the WHERE clause** limit the tuples to which functions are applied, the HAVING clause **serves to choose whole groups.**

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;
```

THE INSERT COMMAND

```
INSERT INTO <table name> [( <column name>{, <column name>} ) ]
(VALUES ( <constant value> , { <constant value>} ){,(<constant value>{,<constant value>})}
| <select statement>)
```

```
INSERT INTO   EMPLOYEE
VALUES        ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
              Oak Forest,Katy,TX', 'M', 37000, '987654321', 4);
```

INSERT is used to add a single tuple to a relation. We must specify the relation name and a list of values for the tuple. The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.

To enter a tuple for a new EMPLOYEE for whom we know only the FNAME, LNAME, DNa, and SSN attributes, we can use

```
INSERT INTO   EMPLOYEE (FNAME, LNAME, DNO, SSN)
VALUES        ('Richard', 'Marini', 4, '653298653');
```

The query would be rejected because no SSN value is provided.

```
INSERT INTO   EMPLOYEE (FNAME, LNAME, DNO)
VALUES        ('Robert', 'Hatcher', 5);
```

THE DELETE COMMAND

```
DELETE FROM <table name>
[WHERE <selection condition>]
```

```
DELETE FROM  EMPLOYEE
WHERE        LNAME='Brown';
DELETE FROM  EMPLOYEE
WHERE        SSN='123456789';
DELETE FROM  EMPLOYEE
WHERE        DNO IN (SELECT   DNUMBER
                     FROM     DEPARTMENT
                     WHERE    DNAME='Research');
DELETE FROM  EMPLOYEE;
```

If applied independently to the database will delete zero, one, four, and all tuples, respectively, from the EMPLOYEE relation.

THE UPDATE COMMAND

```
UPDATE  <table name>
SET  <column name>=<value expression> { , <column name>=<value expression> }
[WHERE  <selection condition>]
```

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

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

DROP COMMAND

```
DROP TABLE  <table name>
```

DROPTABLE DEPENDENT CASCADE;

Ifthe RESTRICT option is chosen instead of CASCADE, a table is dropped only if it is not referenced in any constraints (for example, by foreign key definitions in another relation) or views.

With the CASCADE option, all such constraints and views that reference the table are dropped automatically from the schema, along with the table itself.

One can also drop a schema. For example, if a whole schema is not needed any more, the DROP SCHEMA command can be used. There are two drop behavior options: CASCADE and RESTRICT. For example, to remove the COMPANY database schema and all its tables, domains, and other elements, the CASCADE option is used as follows:

DROP SCHEMA COMPANY CASCADE;

Ifthe RESTRICT option is chosen in place of CASCADE, the schema is dropped only if ithasnoelements in it; otherwise, the DROP command will not be executed.

THE ALTER COMMAND

For example, to add an attribute for keeping track of jobs of employees tothe EMPLOYEE base relations in the COMPANY schema

ALTER TABLE COMPANY.EMPLOYEE ADD JOB VARCHAR(12);

For example, the following command removes the attribute ADDRESS from the EMPLOYEE base table:

ALTER TABLE COMPANY.EMPLOYEE DROP ADDRESS CASCADE;

It is also possible to alter a column definition by dropping an existing default clause or by defining a new default clause

ALTER TABLE COMPANY.DEPARTMENT ALTER MGRSSN DROP DEFAULT;

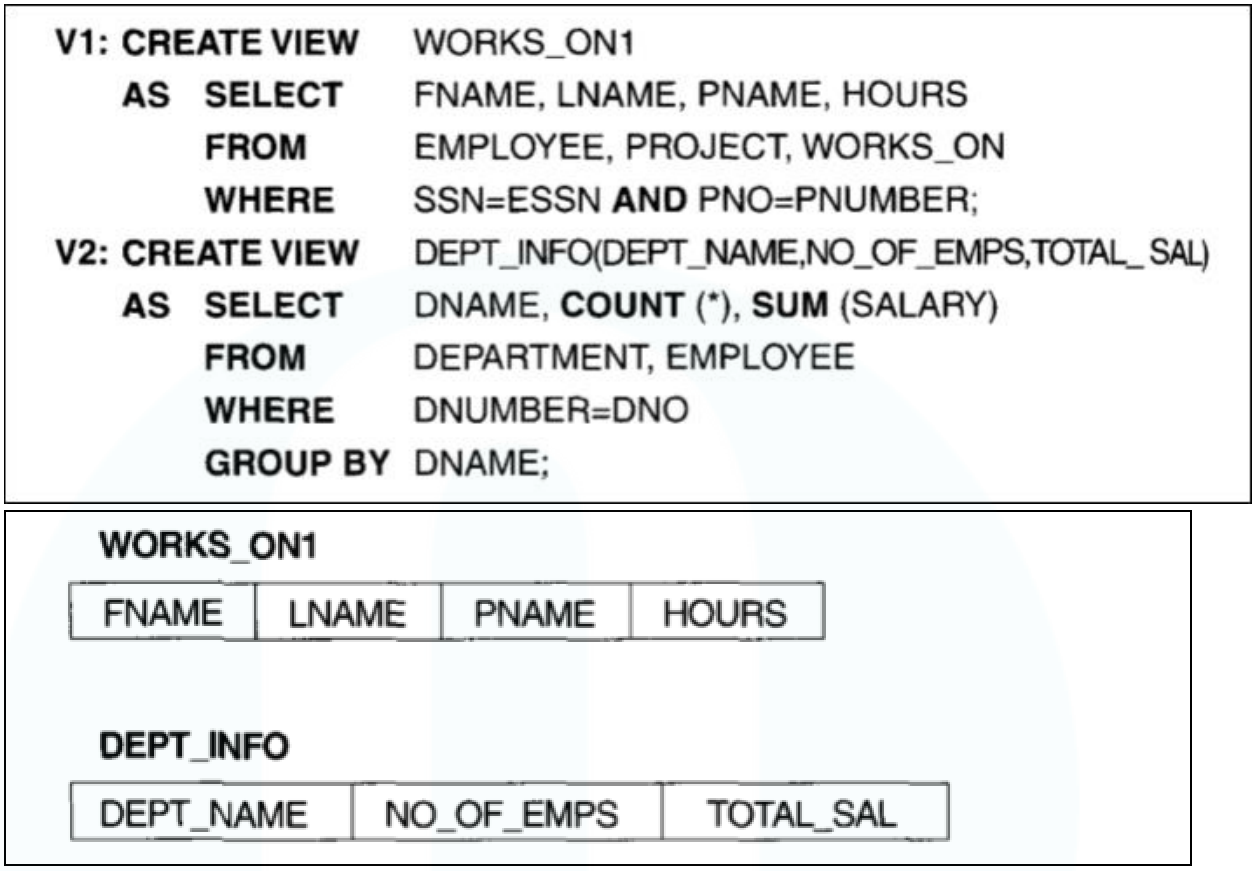
ALTER TABLE COMPANY.DEPARTMENT ALTER MGRSSN SET DEFAULT "333445555";

For example, to drop the constraint named EMPSUPERFK in Figure 8.2 from the EMPLOYEE relation, we write:

ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT EMPSUPERFK CASCADE;

VIEWS

- A view in SQL terminology is a single table that is **derived** from other tables.
- These other tables could be **base tables** or previously defined views.
- Think of a view as a way of specifying a table that we need to **reference frequently**, even though it may not exist physically.
- 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 **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.**



```
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;
```

WORKS_ON1

FNAME	LNAME	PNAME	HOURS

DEPT_INFO

DEPT_NAME	NO_OF_EMPS	TOTAL_SAL

Figure : Two views specified

- In V1, we did not specify any new attribute names for the view WORKS_ONI (although we could have); in this case, WORKS_ONI inherits the names of the view attributes from the defining tables EMPLOYEE, PROJECT, and WORKS_ON.
- View V2 explicitly specifies new attribute names for the view DEPT_INFO, using a one-to-one correspondence between the attributes specified in the CREATE VIEW clause and those specified in the SELECT clause of the query that defines the view.
- 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;

- An efficient strategy for **automatically updating** the **view table** when the **base tables** are **updated** must be developed in order to keep the view up to date.

- Techniques using the concept of ***incremental update*** have been developed for this purpose, where it is determined what new tuples must be inserted, deleted, or modified in a materialized view table when a change is applied to one of the defining base tables.
- The view is generally kept as long as it is being queried.
- If the view is **not queried for a certain period of time**, the system may **then automatically remove the physical view table** and recompute it from scratch when future queries reference the view.
- A **view** with a **single defining table is updatable** if the view attributes contain the primary key of the base relation, as well as all attributes with the NOT NULL constraint that do not have default values specified.
- **Views** defined on multiple tables using **joins** are generally not updatable.
- **Views** defined using **grouping and aggregate functions** are not updatable.

<u>ASSERTIONS</u>

- Users can specify **general constraints**-those that do not fall into any of the categories like primary constraint, referential integrity constraint ,domain constraint **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.
- **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.MGRSSN=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.

- Any WHERE clause condition can be used, but many constraints can be specified using the EXISTS and NOT EXISTS style of SQL conditions.
- The basic technique for writing such assertions is to specify a query that selects any tuples that violate the desired condition.
- By including this query inside a NOT EXISTS clause, the assertion will specify that the result of this query must be **empty**. Thus, the assertion is violated if the result of the query is not empty.
- In our example, the query selects all employees whose salaries are greater than the salary of the manager of their department. If the result of the query is not empty, the assertion is violated.

TRIGGERS

- It may be useful to specify a condition that ,if violated, causes some user to be informed of the violation.
- For example, A manager may want to be informed if an employee's travel expenses exceed a certain limit by receiving a message whenever this occurs.
- The action that the DBMS must take in this case is to send an appropriate message to that user.
- The CREATE TRIGGER statement is used to implement such actions in SQL.

3 COMPONENTS OF TRIGGERS

- event (inserting, changing)-(**before/after**)

These events are specified after the keyword BEFORE , which means that the trigger should be executed before the triggering operation is executed. An alternative is to use AFTER , which specifies that the trigger should be execute after the operation specified in the event is completed.

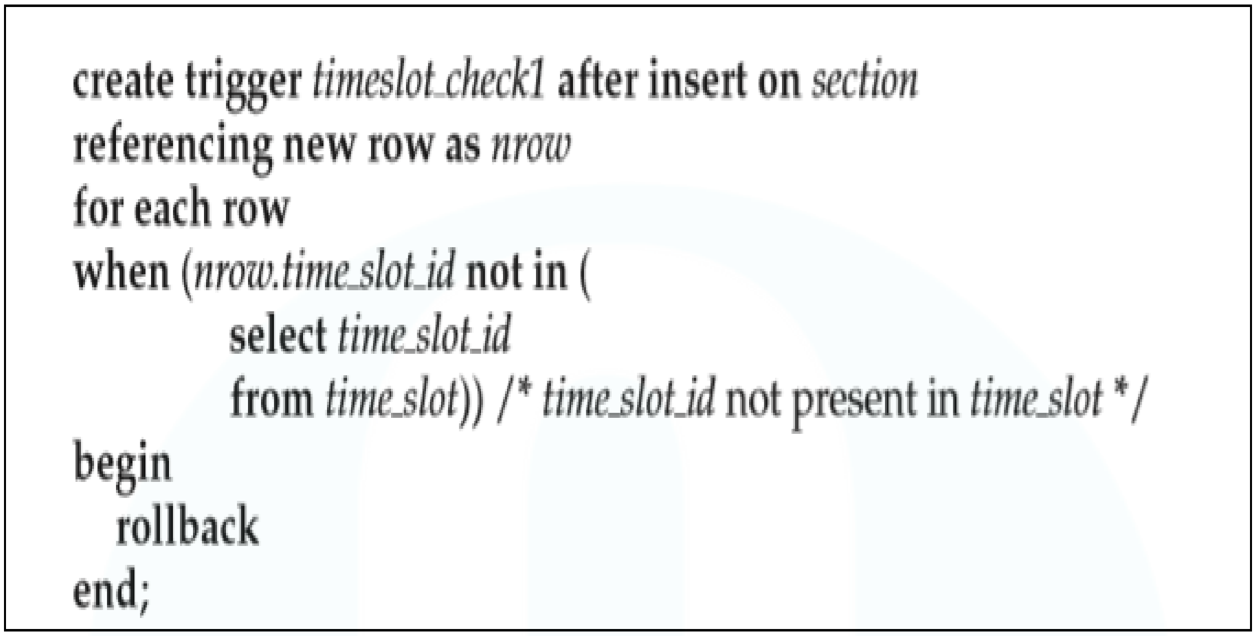
- condition(**when**)

The condition that determines whether the rule action should be executed.Once the triggering event has occurred, an *optional* condition may be evaluated. 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.

- action(**rollback,update**)

The action is usually sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed.

Example 1



```
create trigger timeslot_check1 after insert on section
referencing new row as nrow
for each row
when (nrow.time_slot_id not in (
            select time_slot_id
            from time_slot)) /* time_slot_id not present in time_slot */
begin
  rollback
end;
```

Timeslot_check1 is the name of the TRIGGER

Section and time_slot are the 2 TABLES . Both tables contain time_slot_id as the ATTRIBUTES.

Time_slot_id act as foreign key in the table *section* whereas primary key in the table *time_slot*

Trigger gets activates when foreign key(time_slot_id) in *section* tries to enter values not in primary key (time_slot_id) of time_slot since it violates referential integrity. If it violates ,rollback happens.

The first trigger definition in the figure specifies that the trigger is initiated after any insert on the relation section and it ensures that the time slot id value being inserted is valid.

An SQL insert statement could insert multiple tuples of the relation, and **the for each row** clause in the trigger code would then explicitly iterate over each inserted row.

The **referencing new row as** clause creates a variable **nrow**(called a transition variable)that **stores** the **value of an inserted row after the insertion**.

The **when** statement specifies a condition. The system executes the rest of the trigger body only for tuples that satisfy the condition.

Example 2

```
create trigger timeslot_check2 after delete on timeslot
referencing old row as orow
for each row
when (orow.time_slot_id not in (
            select time_slot_id
            from time_slot) /* last tuple for time_slot_id deleted from time_slot */
   and orow.time_slot_id in (
            select time_slot_id
            from section)) /* and time_slot_id still referenced from section*/
begin
   rollback
end;
```

Timeslot_check2 is the name of the TRIGGER

Section and time_slot are the 2 TABLES . Both tables contain time_slot_id as the ATTRIBUTES.

Time_slot_id act as foreign key in the table *section* whereas primary key in the table *time_slot*

Trigger gets activates when primary key (time_slot_id) of time_slot tries to delete some of its value and foreign key(time_slot_id) of *section* still have references for this deleted values in its table(*section*),because it violates referential integrity .If it violates ,rollback happens.

Example 3

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> 'F' and nrow.grade is not null
   and (orow.grade = 'F' or orow.grade is null)
begin atomic
   update student
   set tot_cred= tot_cred+
           (select credits
            from course
            where course.course_id= nrow.course_id)
   where student.id = nrow.id;
end;
```

Table→takes, student

Attribute→grade

Trigger activates when *grade* gets updated, which in turn updates the total credits.