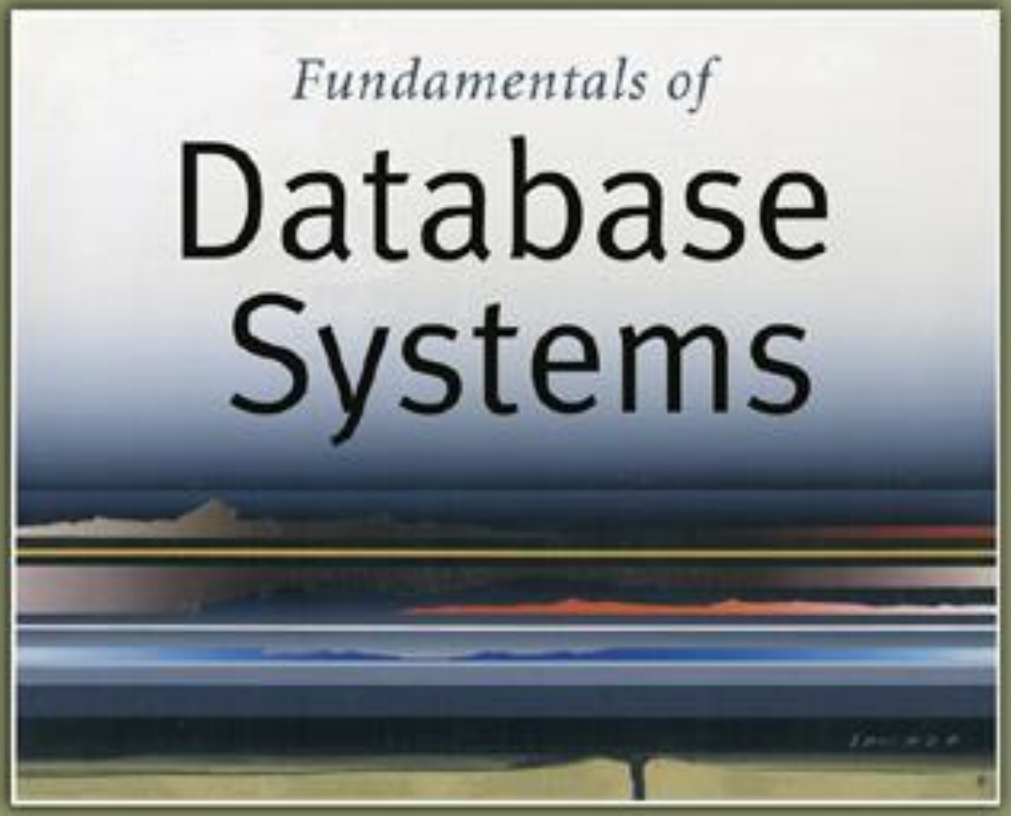
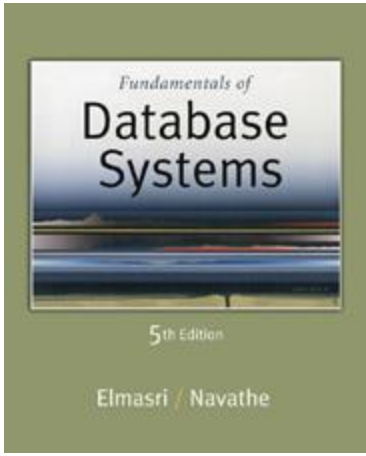


5th Edition

Elmasri / Navathe

Chapter 8

SQL-99: SchemaDefinition, Constraints, and Queries and Views



sQL

- Structured Query Language
- SQL is a computer language for storing, Manipulating and retrieving data stored in relational database
- SQL is the standard language for Relation Database System. All relational database management systems like “ MYSQL, MS Acess, Oracle, Sybase, Informix, postgres and SQL Server” Use SQL as standard database language

Relational Database Schema-Company Dtabase

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

Cont..

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

SQL Data Definition and Data Types

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, types, constraints, views, domains, and other constructs (such as authorization grants) that describe the schema. The general form

CREATE SCHEMA schema-name **AUTHORIZATION** identifier;

For Example

CREATE SCHEMA COMPANY **AUTHORIZATION** 'Jsmith';

Create Table Command

- The **CREATE TABLE** command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints.
- The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and possibly attribute constraints, such as NOT NULL.
- The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement after the attributes are declared, or they can be added later using the ALTER TABLE
- The general form is

Cont..

```
CREATE TABLE Table-name
(
      Attribute1 datatype Constraint,
      Attribute2 datatype Constraint,
      Attribute3 datatype Constraint,
        …
        …
      PRIMARY KEY (attribute),
      FOREIGN KEY (attribute) REFERENCES attribute
);
```

Attribute Data Types and Domains in SQL

- The basic **data types** available for attributes include numeric, character string, bit string, Boolean, date, and time.
- **Numeric** data types include integer numbers of various sizes. (INTEGER or INT, and SMALLINT) and 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*)

- 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.

- **Character-string** data types are 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.

- A **Boolean** data type has the traditional values of TRUE or FALSE.

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

DROP TABLE

- The DROP command can be used to drop *named* schema elements, such as tables, domains, types, or constraints
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- 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**;

- If the RESTRICT option is chosen in place of CASCADE, the schema is dropped only if it has *no elements* in it; otherwise, the DROP command will not be executed.

- If a base relation within a schema is no longer needed, the relation and its definition

  can be deleted by using the DROP TABLE command.

  DROP TABLE DEPENDENT **CASCADE**;

- If the 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 or by any other elements

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 (contd.)

- 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');
```

Retrieval Queries in SQL (contd.)

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

Simple SQL Queries (contd.)

- Example of a simple query on one relation
- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith‘.

Cont..

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

Simple SQL Queries (contd.)

- 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;
```

Simple SQL Queries (contd.)

- 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.

Cont..

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

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
- Example:

- **EMPLOYEE.**LNAME, **DEPARTMENT.**DNAME

Cont..

SELECT EMPLOYEE.Fname, EMPLOYEE.LName, EMPLOYEE.Address

FROM EMPLOYEE, DEPARTMENT

WHERE DEPARTMENT.DName = 'Research' **AND**

DEPARTMENT.Dnumber = EMPLOYEE.Dno;

ALIASES

- Some queries need to refer to the same relation twice
  - In this case, *aliases* are given to the relation name
  - Can also use the AS keyword to specify aliases

- 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 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
- Query 9: 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 (contd.)

- 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
```

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

Relational Algebra Operations from Set Theory: UNION

- UNION Operation
  - Binary operation, denoted by $\cup$
  - The result of R $\cup$ S, is a relation that includes all tuples that are either in R or in S or in both R and S
  - Duplicate tuples are eliminated
  - The two operand relations R and S must be "type compatible" (or UNION compatible)
    - R and S must have same number of attributes
    - Each pair of corresponding attributes must be type compatible (have same or compatible domains)

Example of the result of a UNION operation

- UNION Example

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Relational Algebra Operations from Set Theory: INTERSECTION

- INTERSECTION is denoted by ∩
- The result of the operation R ∩ S, is a relation that includes all tuples that are in both R and S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be "type compatible"

Relational Algebra Operations from Set Theory: SET DIFFERENCE (cont.)

- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of R – S, is a relation that includes all tuples that are in R but not in S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be "type compatible"

Example to illustrate the result of UNION, INTERSECT, and DIFFERENCE

(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

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.

SET OPERATIONS (contd.)

- Query 4: 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')
```

SUBSTRING COMPARISON

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

SUBSTRING COMPARISON (contd.)

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

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

SUBSTRING COMPARISON (contd.)

- Query : 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

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 : Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

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

ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- 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
- The general format

  ORDER BY Attribute-name ASC;

ORDER BY (contd.)

- Query : 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;
```

Cont..

The general form is

SELECT <attribute list>

FROM <table list>

[**WHERE** <condition>]

[**ORDER BY** <attribute list>];

NESTING OF QUERIES

- A complete SELECT query, called a *nested query*, can be specified within the WHERE-clause of another query, called the *outer query*
  - Many of the previous queries can be specified in an alternative form using nesting

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

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

NESTING OF QUERIES (contd.)

- 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
- v 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)
```

Cont..

- SELECT E.FNAME, E.LNAME
  FROM EMPLOYEE AS E, DEPENDENT
  WHERE ESSN= E.SSN AND
  E.FNAME= DEPENDENT_NAME;

QUERY : **select the Essns of all employees who work the same (project, hours) combination on some project that employee 'John Smith' (whose Ssn = '123456789') works on.**

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

Cont..

The comparison condition (v > ALL V) returns TRUE if the value v is greater than *all* the values in the set (or multiset) V.

QUERY : which returns the names of employees whose salary is greater than the salary of all the employees in department 5

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

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not

THE EXISTS FUNCTION (contd.)

- 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);
```

THE EXISTS FUNCTION (contd.)

- 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
  - Relation1 JOIN Relation2 ON condition
  - Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

Joined Relations Feature in SQL2 (contd.)

- Examples:

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

- 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'
```

AGGREGATE FUNCTIONS

- Include **COUNT, SUM, MAX, MIN, and AVG**
- Query 15: 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

AGGREGATE FUNCTIONS (contd.)

- 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 (contd.)

- 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'
```

GROUPING

- 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*
- *GROUP BY Grouping attribute*

GROUPING (contd.)

- Query 20: 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 (contd.)

- Query 21: 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

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 general form is:

Having (Condition);

THE HAVING-CLAUSE (contd.)

- 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;
```

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 (contd.)

- 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

BETWEEN Clause

- The general form

  Attribute name BETWEEN range;

- Retrieve all employees in department number 5 whose salary between $30000 and $40000.

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

ALTER table command

he SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table.
You should also use the ALTER TABLE command to add and drop various constraints on an existing table.

Syntax

The basic syntax of an ALTER TABLE command to add a **New Column** in an existing table is as follows.

```
ALTER TABLE table_name ADD column_name datatype;
```

The basic syntax of an ALTER TABLE command to **DROP COLUMN** in an existing table is as follows.

```
ALTER TABLE table_name DROP COLUMN column_name;
```

The basic syntax of an ALTER TABLE command to change the **DATA TYPE** of a column in a table is as follows.

```
ALTER TABLE table_name MODIFY COLUMN column_name datatype;
```

The basic syntax of an ALTER TABLE command to add a **NOT NULL** constraint to a column in a table is as follows.

```
ALTER TABLE table_name MODIFY column_name datatype NOT NULL;
```

The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows.

```
ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
```

The basic syntax of an ALTER TABLE command to **ADD CHECK CONSTRAINT** to a table is as follows.

```
ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);
```

The basic syntax of an ALTER TABLE command to **ADD PRIMARY KEY** constraint to a table is as follows.

```
ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
```

The basic syntax of an ALTER TABLE command to **DROP CONSTRAINT** from a table is as follows.

```
ALTER TABLE table_name DROP CONSTRAINT MyUniqueConstraint;
```

If you're using MySQL, the code is as follows −

```
ALTER TABLE table_name DROP INDEX MyUniqueConstraint;
```

The basic syntax of an ALTER TABLE command to **DROP PRIMARY KEY** constraint from a table is as follows.

```
ALTER TABLE table_name DROP CONSTRAINT MyPrimaryKey;
```

If you're using MySQL, the code is as follows −

```
ALTER TABLE table_name DROP PRIMARY KEY;
```

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 (contd.)

- 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 (contd.)

- 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 (contd.)

- 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 (contd.)

- 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
  - Referential integrity should be enforced
  - 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

DELETE (contd.)

- 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 (contd.)

The general form is

```
UPDATE    Table-name
          SET       new values
          WHERE     condition;
```

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

U5:

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

UPDATE (contd.)

- 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

Recap 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>]
```

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

ALTER TABLE

- Used to 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
- Example:

```
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.