SQL developments: an overview

- In 1986, ANSI and ISO published an initial standard for SQL: SQL-86 or SQL1
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL-92
- In 1999, SQL-99 (SQL3) was released with support for object-oriented data management
- In late 2003, SQL-2003 was released
- Now: SQL-2006 was published

Basic SQL

- SQL Data Definition & Data Types
- Specifying Constraints in SQL
- Basic Retrieval Queries in SQL
- INSERT, DELETE, UPDATE

SQL

- *DDL: Create*, Alter, Drop
- *DML: Select, Insert, Update, Delete*
- DCL: Commit, Rollback, Grant, Revoke

CREATE SCHEMA

- Started with SQL 92
- A SQL Schema: is to group together tables and other constructs that belong to the same database application

CREATE SCHEMA SchemaName
AUTHORIZATION AuthorizationIdentifier

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

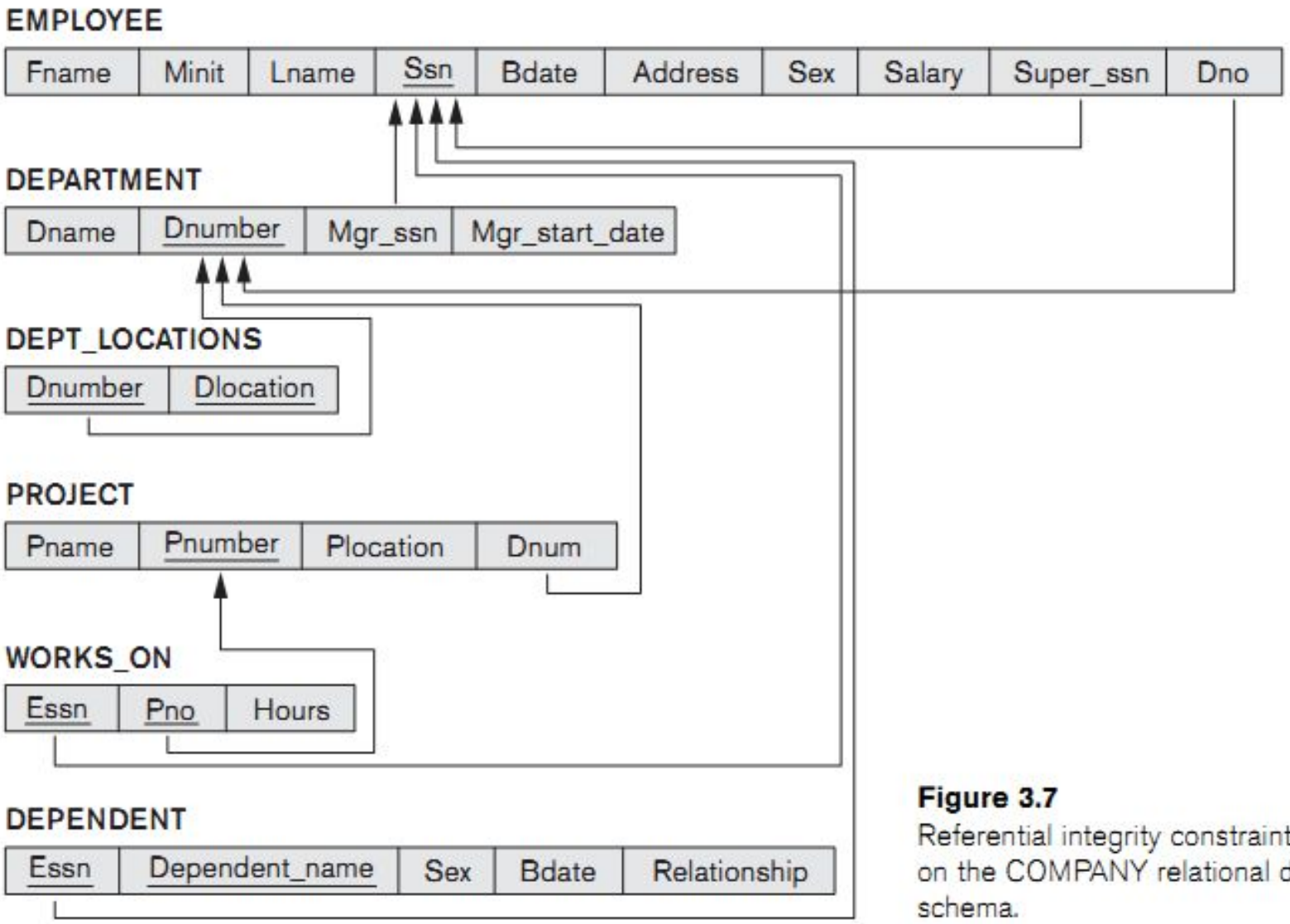


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

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

- CREATE TABLE Company.TableName …

or

- CREATE TABLE TableName …

```
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),
          Super_ssn      CHAR(9),
          Dno            INT                NOT NULL,
        PRIMARY KEY (Ssn),
```

```
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)         NOT NULL,
          Dnumber               INT                 NOT NULL,
          Mgr_ssn               CHAR(9)             NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

```
CREATE TABLE DEPT_LOCATIONS
       ( Dnumber          INT             NOT NULL,
         Dlocation        VARCHAR(15)     NOT NULL,
       PRIMARY KEY (Dnumber, Dlocation),
       FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

```
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)         NOT NULL,
          Pnumber               INT                 NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                 NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
```

```
CREATE TABLE WORKS_ON
    ( Essn          CHAR(9)         NOT NULL,
      Pno           INT             NOT NULL,
      Hours         DECIMAL(3,1)    NOT NULL,
    PRIMARY KEY (Essn, Pno),
    FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
    FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
```

```
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) );
```

Data Types

- **Numeric:** INT or INTEGER, FLOAT or REAL, DOUBLE PRECISION, …
- **Character string:** fixed length CHAR(n), varying length VARCHAR(n)
- **Bit string:** BIT(n), e.g. B'1001'
- **Boolean:** true, false or NULL
- **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

Data Types

- A domain can be declared and used with the attribute specification

CREATE DOMAIN DomainName AS DataType [CHECK conditions];

Example:

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);
```

CREATE TABLE

```
CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] […,]}
{[FOREIGN KEY (listOfFKColumns)
  REFERENCES ParentTableName [(listOfCKColumns)],
  [ON UPDATE referentialAction]
  [ON DELETE referentialAction ]] [,…]}
{[CHECK (searchCondition)] [,…] })
```

Specifying Constraints in SQL

- Specifying Attribute Constraints and Attribute Defaults
- Default values
  - DEFAULT <value> can be specified for an attribute
  - If no default clause is specified, the default value is NULL for attributes that do not have the NOT NULL constraint
- CHECK clause: restrict attribute or domain values

  DNUMBER INT NOT NULL CHECK (DNUMBER>0 AND DNUMBER<21);
  - CREATE DOMAIN can also be used in conjunction with the CHECK clause:

  CREATE DOMAIN D_NUM AS INTEGER CHECK (D_NUM>0 AND D_NUM<21);

```
CREATE TABLE EMPLOYEE
( … ,
Dno INT NOT NULL DEFAULT 1,
CONSTRAINT EMPPK
PRIMARY KEY (Ssn),
CONSTRAINT EMPSUPERFK
FOREIGN KEY (Super_ssn) 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);
```

```
CREATE TABLE DEPARTMENT
( … ,
Mgr_ssn CHAR(9) NOT NULL DEFAULT '888665555',
… ,
CONSTRAINT DEPTPK
PRIMARY KEY(Dnumber),
CONSTRAINT DEPTSK
UNIQUE (Dname),
CONSTRAINT DEPTMGRFK
FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
ON DELETE SET DEFAULT ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
( … ,
PRIMARY KEY (Dnumber, Dlocation),
FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
ON DELETE CASCADE ON UPDATE CASCADE);
```

Specifying Constraints in SQL

- Specifying Key Constraints
- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases

```
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  );
```

Or **Dnumber INTEGER PRIMARY KEY;**

REFERENTIAL INTEGRITY OPTIONS

- Specifying Referential Integrity Constraints: FOREIGN KEY clause. Can specify RESTRICT, CASCADE, SET NULL or SET DEFAULT on referential integrity constraints

```
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
);
```

Specifying Constraints in SQL

- Giving names to constraints

```
CREATE TABLE EMPLOYEE
    (  . . . ,
       Dno         INT             NOT NULL        DEFAULT 1,
    CONSTRAINT EMPPK
       PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
       FOREIGN KEY (Super_ssn) 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);
```

Specifying Constraints in SQL

- Specifying Constraints on Tuples (tuple-based) using CHECK: at the end of CREATE TABLE
- Example:

```
CHECK (Dept_create_date <= Mgr_start_date);
```

Schema Change Statements in SQL

- The DROP command

DROP SCHEMA COMPANY CASCADE;

DROP TABLE DEPENDENT CASCADE;

The ALTER command

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

ALTER TABLE COMPANY.EMPLOYEE
DROP COLUMN ADDRESS CASCADE;

ALTER TABLE COMPANY.DEPARTMENT **ALTER COLUMN** MGR_SSN **DROP DEFAULT;**

ALTER TABLE COMPANY.DEPARTMENT **ALTER COLUMN** MGR_SSN **SET DEFAULT** '333445555'

ALTER TABLE COMPANY.EMPLOYEE **DROP CONSTRAINT** EMPSUPERFK **CASCADE;**

Basic Retrieval Queries in SQL

- SELECT statement
- 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

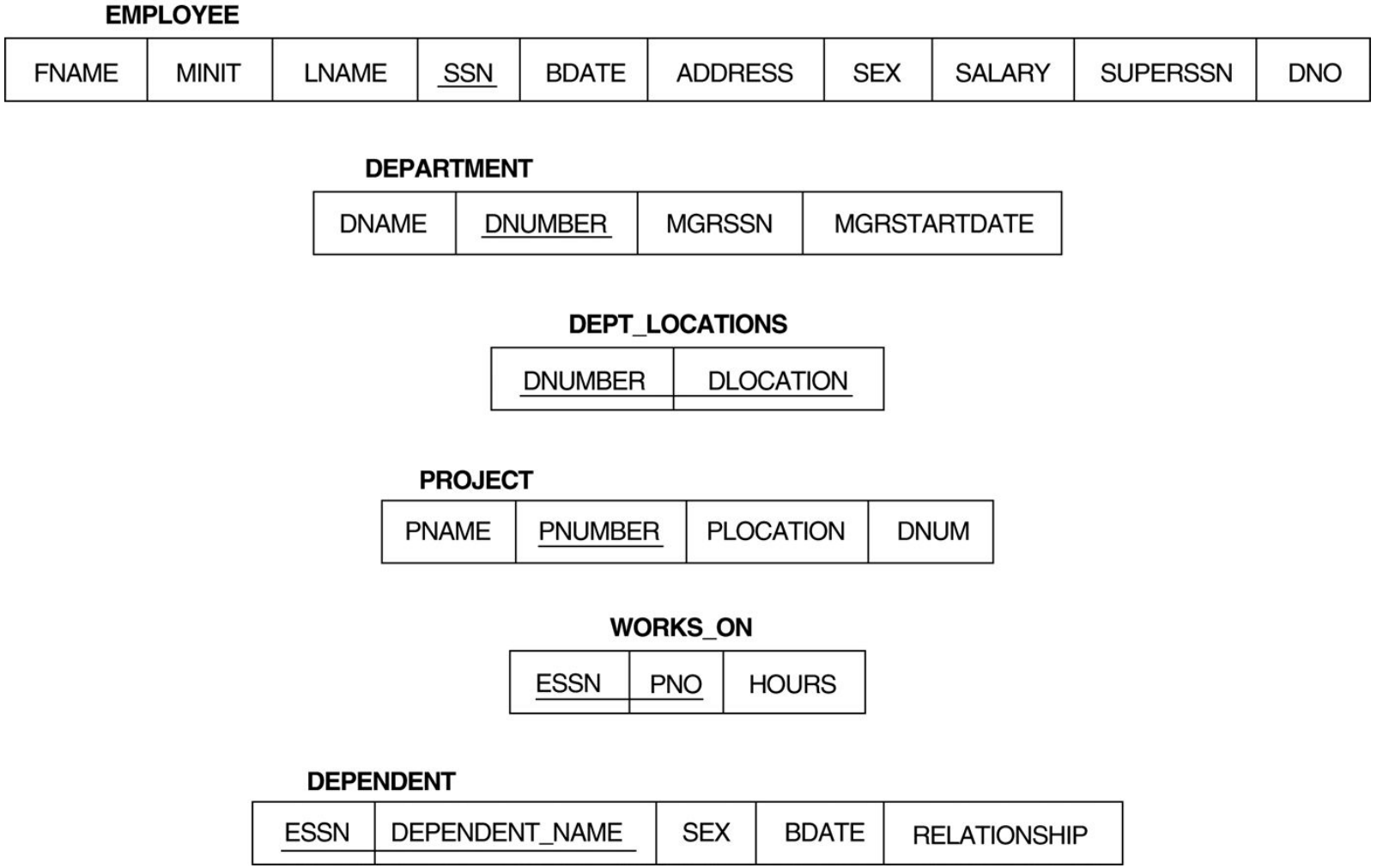
Basic Retrieval Queries in SQL (cont.)

- 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



Populated Database

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 Comparison Operators:

Operator	Description
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

SQL Logical Operators:

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list as per the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.

Contd:

LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.

Simple SQL Queries

- 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'
```

  - The SELECT-clause specifies the projection attributes and the WHERE-clause specifies the selection condition
  - 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
```

  - (DNAME='Research') is a ***selection condition***
  - (DNUMBER=DNO) is a ***join condition***

Figure 4.3
Results of SQL queries when applied to the COMPANY database state shown in Figure 3.6. (a) Q0. (b) Q1. (c) Q2. (d) Q8. (e) Q9. (f) Q10. (g) Q1C.

(a)

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

(b)

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

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

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

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 (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

(c)

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

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

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

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

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

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

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

(d)

E.Fname	E.Lname	S.Fname	S.Lname
John	Smith	Franklin	Wong
Franklin	Wong	James	Borg
Alicia	Zelaya	Jennifer	Wallace
Jennifer	Wallace	James	Borg
Ramesh	Narayan	Franklin	Wong
Joyce	English	Franklin	Wong
Ahmad	Jabbar	Jennifer	Wallace

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

Q9: SELECT SSN
FROM EMPLOYEE

(e)

SSN
123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555

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

- **SELECT SSN, DNAME**
  FROM EMPLOYEE, DEPARTMENT

(f)

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

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

Q1C:

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

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-09-01	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

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

```
SELECT   SALARY
FROM         EMPLOYEE;

SELECT   DISTINCT SALARY
FROM         EMPLOYEE;

SELECT   ALL SALARY
FROM         EMPLOYEE;
```

(a)

Salary
30000
40000
25000
43000
38000
25000
25000
55000

(b)

Salary
30000
40000
25000
43000
38000
55000

SUBSTRING COMPARISON OPERATORS:

The following table has a few examples showing the WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200.
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position.
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions.
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length.
WHERE SALARY LIKE '%2'	Finds any values that end with 2.
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3.
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3.

SUBSTRING COMPARISON

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

SUBSTRING COMPARISON (cont.)

- 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.
- 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 '/‘ 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’

Query 14. 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;
```

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

Ordering of Query Results

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

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

- ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC

More SQL: Complex Queries

Comparisons Involving NULL and Three-Valued Logic:

Unknown value

Unavailable or withheld value.

Not applicable attribute.

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Query 18. Retrieve the names of all employees who do not have supervisors.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Super_ssn IS NULL;
```

Nested Queries, Tuples, and Set/Multiset Comparisons

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

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

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

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

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

Aggregate Functions in SQL

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

the minimum salary, and the average salary.

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

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

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

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

Q21: SELECT COUNT (*)

FROM EMPLOYEE;

Q22: SELECT COUNT (*)

FROM EMPLOYEE, DEPARTMENT

WHERE DNO = DNUMBER AND DNAME = 'Research';

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

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

To retrieve the names of all employees who have two or more dependents (Query 5), we can write the following:

Q5: SELECT Lname, Fname

FROM EMPLOYEE

WHERE (SELECT COUNT (*)

FROM DEPENDENT

WHERE Ssn = Essn) > = 2;

Grouping: The GROUP BY and HAVING Clauses

Query 24. 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;
```

Results of GROUP BY and HAVING. (a) Q24. (b) Q26.

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	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

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

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

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

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

(b)

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.

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)

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

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

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
SELECT Dno, COUNT (*)
FROM EMPLOYEE
WHERE Salary>40000 AND Dno IN
( SELECT Dno
FROM EMPLOYEE
GROUP BY Dno
HAVING COUNT (*) > 5)
GROUP BY Dno;
```

Specifying Updates in SQL

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

INSERT

- 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 INTODEPTS_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

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>]

Query 4. 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 Mgr_ssn = Ssn
AND Lname = 'Smith' )
UNION
( SELECT DISTINCT Pnumber
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE Pnumber = Pno AND Essn = Ssn
AND Lname = 'Smith' );
```