SQL-99

Schema Definition

Basic Constraints

Queries

SQL - The Relational Database Standard

- SQL → Structured Query Language
  - A high-level declarative language interface.
  - includes some features from relational algebra
  - ***It is based to a greater extent on the tuple relational calculus***
- SQL is a comprehensive database language. it include
  - statements for data definition,
  - statements for query,
  - statements for update.

[it is both a DDL (Data Definition Language) and a DML (Data Manipulation Language).]

SQL - The Relational Database Standard

- In addition, it has facilities for
  - defining views on the database
  - specifying security and authorization
  - defining integrity constraints
  - specifying transaction controls

- Allows embedding SQL statements into a general-purpose programming language(C ,C++, java ,Visual basic ...)

Topics to be discussed

1. SQL Data Definition and Data Types
2. Specifying Basic Constraints in SQL
3. Schema Change Statements
4. Basic Queries in SQL
5. More Complex Queries
6. Insert , delete and update statements in SQL
7. Additional features of SQL
8. Views (Virtual Tables) in SQL
9. Database Programming : Issues and Techniques
10. Embedded SQL

SQL DATA DEFINITION AND DATA TYPES

SQL Data Definition

- In SQL terminology ,
  - A relation is called a **table**
  - A tuple is called a **row**
  - An attribute is called a **column**
- **CREATE** statement : The main SQL command for data definition.

  It can used to create
  - **schemas**
  - **tables**
  - **domains**
  - **views**
  - **Assertion**
  - **triggers**

SQL Data Definition : Schema and Catalog Concepts

An SQL schema is identified by a **schema name**, and includes

- an **authorization identifier** to indicate the user or account who owns the schema
- **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

SQL Data Definition : Schema and Catalog Concepts

- A schema is created via the **CREATE SCHEMA** statement, which can include all the schema elements' definitions.

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 schema and schema elements .

The privilege for these must be given by Database Administrator (DBA)

SQL Data Definition : Schema and Catalog Concepts

SQL catalog

A named collection of schemas in an SQL environment.

- A catalog always contains a special schema called I**NFORMATION_SCHEMA**
- I**NFORMATION_SCHEMA** provides information on all the element descriptors of all the schemas in the catalog to authorized users.

SQL Data Definition : The CREATE TABLE Command

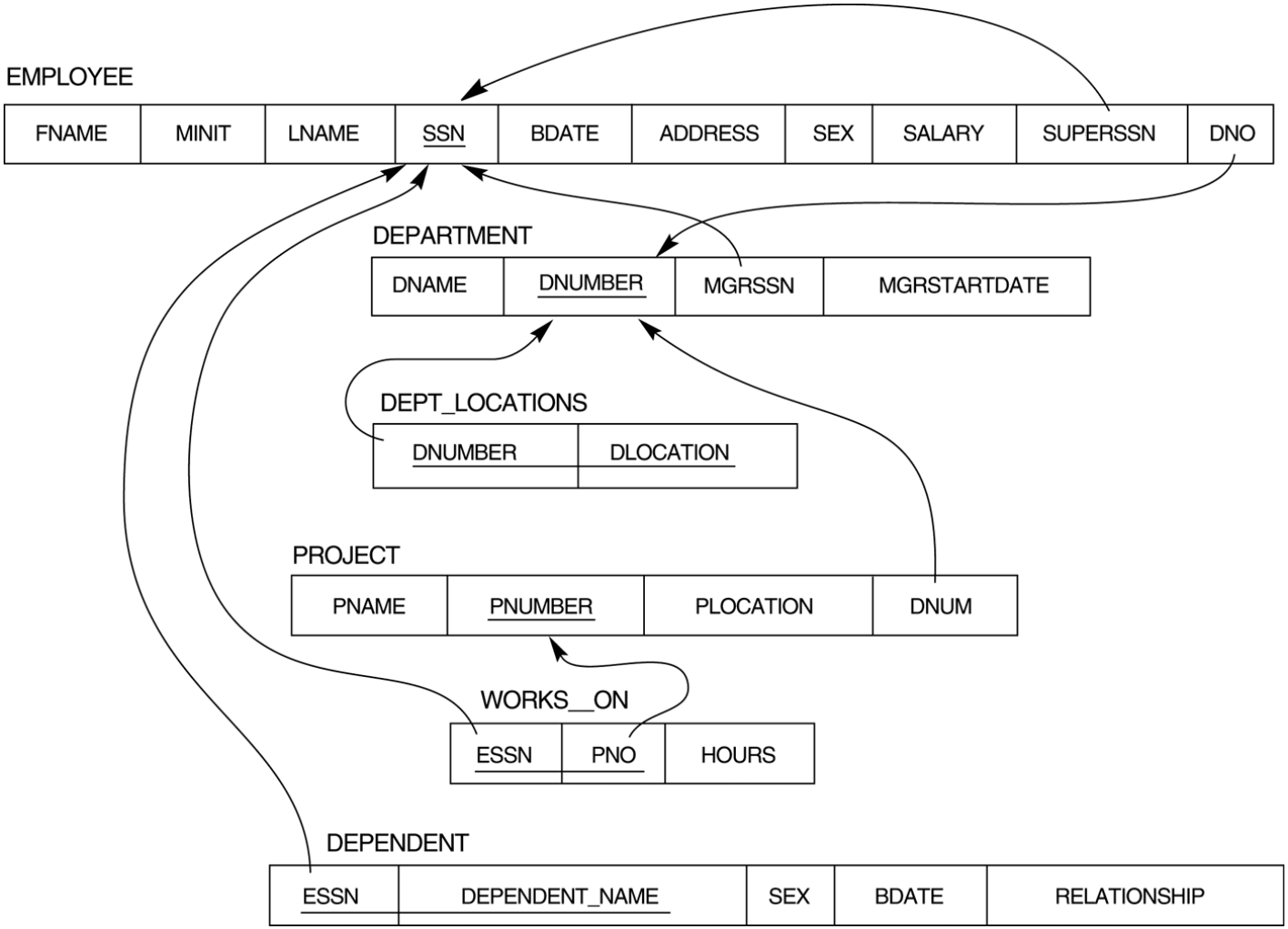
- Used to specify a new table by giving it a name and specifying its attributes and constraints
  - The **attributes** are specified first, and each attribute is given a name
  - Followed by a **data type** to specify its domain of values.
  - Followed by any attribute constraints such as NOT NULL.
  - The key, entity integrity, and referential integrity constraints can be specified—within the CREATE TABLE statement—after the attributes are declared.
- **Syntax :**

CREATE TABLE COMPANY.EMPLOYEE… **OR**

CREATE TABLE EMPLOYEE **…**

SQL Data Definition : The CREATE TABLE Command

Example : Company Database Schema



SQL Data Definition : The CREATE TABLE Command

Example : Company Database

```
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) ) ;
```

SQL Data Definition : The CREATE TABLE Command

Example : Company Database

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

SQL Data Definition : The CREATE TABLE Command

Example : Company Database

```
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) ) ;
```

SQL Data Definition : The CREATE TABLE Command

Example : Company Database

```
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) ) ;
```

SQL Data Definition : The CREATE Domain Command

Create Domain and use with attribute specification

CREATE DOMAIN SSN_TYPE AS CHAR(9)

Use SSS_TYPE in the place of CHAR(9)

Advantage :

- Easier to change data type for domain
- Improves readability

SQL Data Definition : The CREATE TABLE Command

- The relations declared through CREATE TABLE statements are called **base tables** (or base relations)

- The relations declared through CREATE VIEW statements are called **virtual tables** (or virtual relations)

- In SQL the attributes in a base relation are considered to be *ordered in the sequence* in which they are specified in the CREATE table statement.

- Rows are not considered to be ordered within a relation

SQL DATA Types

- **Numeric data types**
  - Integer numbers of various sizes

    INTERGER or INT and SMALLINT
  - Floating points of various precision

    FLOAT or REAL

    DOUBLE PRECISION
  - Formatted numbers can be declared by using
    - DECIMAL (i , j) or DEC(i , j) or NUMERIC(i , j) where i is the precision and j is the scale
    - The default scale is zero and default for precision is implementation defined.

SQL DATA Types

- **Character-string →2 types**

  1. **Fixed-length**

     CHAR(*n*) or CHARACTER(*n*), where *n* is the number of characters.

  2. **Varying-length**

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

- Literal string value is placed between single quotation marks and it is case sensitive.

  Example **'SMITH'** and **'smith'**

SQL DATA Types

- **Bit-string → 2 types**

    1. **Fixed length**

        BIT(*n*), where *n* is the maximum number of bits.

    2. **Varying length**

        BIT VARYING(*n*), where *n* is the maximum number of bits.

    The default for *n,* the length of a character string or bit string, is one.

- Literal bit strings are placed between single quotes but precede by a B ;

    Example B’10101’

SQL DATA Types

- **Boolean**
  - **Traditional Values: TRUE or FALSE**
  - **Third possible values : UNKNOWN**

TABLE 8.1 LOGICAL CONNECTIVES IN THREE-VALUED LOGIC

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

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

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

SQL DATA Types

- **Date and Time**
  - The DATE data type has ten positions

    Typically form **YYYY-MM-DD.**

  - The TIME data type has at least eight positions

    Typically form **HH:MM:SS.**

  - Literal values are represented by single quoted strings preceded by DATE or TIME

    DATE'2002-09-27' or **TIME'09:12:47'**

  - The **< (less than)** comparison can be used with dates or times.

SQL DATA Types

- **Timestamp**
  - A timestamp data type (TIMESTAMP) includes both the DATE and TIME fields, plus a maximum number of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier
  - Literal values are specified as follows:

    TIMESTAMP'2002 -09-27 09:12:47 648302'

    (Minimum one space between date and time)

SQL DATA Types

- **Interval**
  - INTERVAL data type specifies an **interval**—a *relative value* that can be used to increment or decrement an absolute value of a date, time, or timestamp.

Specifying Basic Constraints in SQL-99

Specifying Not Null, DEFAULT and CHECK constraints

1. Not Null constraints :

- This should always be specified for the primary key attributes of each relation,
- Also for any other attributes whose values are required not to be NULL.

Example : DNAME VARCHAR(20) NOT NULL

Specifying Not Null, DEFAULT and CHECK constraints

2. DEFAULT Constraint:

- To define a default value for an attribute

Example : DNO INT NOT NULL DEFAULT 1

3. CHECK Constraint :

- To restrict the attribute or domain values.

Example :

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

Specifying Key and Referential Integrity Constraints:

1. The PRIMARY KEY clause

- Specifies one or more attributes that make up the primary key of a relation.

PRIMARY KEY (DNUMBER,DLOCATION)

- If primary key single attribute then the clause can follow the attribute directly

DNUMBER INT PRIMARY KEY

Specifying Key and Referential Integrity Constraints:

2. The UNIQUE clause

- Specifies alternate (or secondary) keys.

UNIQUE (DNAME)

3. FOREIGN KEY clause.

- Specifies referential integrity constraint

FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER)

Specifying Key and Referential Integrity Constraints:

- **Options to avoid violations**

1. The options include SET NULL, CASCADE, and SET DEFAULT.
2. An option must be qualified with either ON DELETE or ON UPDATE.
3. The action for CASCADE ON DELETE is to delete all the referencing tuples,
4. 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.

Example:

FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER)
ON DELETE SET DEFAULT ON UPDATE CASCADE.

Specifying Key and Referential Integrity Constraints:

- **Giving Names to Constraints**

  - **Constraints** may be given a name, following the keyword CONSTRAINT.

Example :

CONSTRAINT *DEPTPK* PRIMARY KEY(DNUMBER)

- The names all the constraints must be unique within a schema.
- Constraints can be dropped or replaced with another constraint.
- Giving names to constraints is optional.

Specifying Constraints on Tuples Using CHECK

- These are called tuple based constraints which are specified at the end of a CREATE TABLE statement.

Example : **CHECK (DEPT_CREATE_DATE <MGRSTARTDATE);**

Because a manager can start managing a department only after it is created

Example : Company Database

```
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 );
```

Example : Company Database

```
CREATE TABLE DEPARTMENT
        ( . . . ,
          MGRSSN    CHAR(9)  NOT NULL DEFAULT  '888665555' ,
          . . . ,
          CONSTRAINT DEPTPK
            PRIMARY KEY (DNUMBER) ,
          CONSTRAINT DEPTSK
            UNIQUE (DNAME),
          CONSTRAINT DEPTMGRFK
            FOREIGN KEY (MGRSSN) 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 ) ;
```

Schema Change Statement in SQL-99

The DROP Command

- **To drop**
  - schema
  - tables
  - domains
  - constraints.
- **DROP has two options : CASCADE & RESTRICT**

Example : DROP SCHEMA COMPANY **CASCADE;**

- **CASCADE** option , when dropping a schema , will remove database schema and all its tables, domains, and other elements.
- **If RESTRICT** option is chosen ,then schema is dropped only if it has no elements in it; otherwise, the DROP command will not be executed

The DROP Command

Example : DROP TABLE DEPEDENT CASCADE;

- **CASCADE** option , when dropping a table , will remove table and all its definitions (including constraints)
- **RESTRICT** option is chosen, the table is dropped only if it is not referenced in any constraints or views . Otherwise DROP command will not be executed.

The ALTER TABLE Command

- To change the definition of a base table

- **ALTER TABLE is used to**

    1. add a column (attribute)
    2. drop a column (attribute)
    3. change column definition
    4. add or dropping table constraints.

The ALTER TABLE Command

Examples:

1. `ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);`
2. `ALTER TABLE EMPLOYEE DROP ADDRESS CASCADE;`
3. `ALTER TABLE DEPARTMENT ALTER MGRSSN DROP DEFAULT;`
4. `ALTER TABLE DEPARTMENT ALTER MGRSSN SET DEFAULT "333445555";`
5. `ALTER TABLE EMPLOYE DROP CONSTRAINT EMPSUPERFK CASCADE;`

Basic Queries in SQL -99

The SELECT-FROM-WHERE

- **For retrieving information from a database**
- ***An SQL table is not a set of tuples. So any 2 tuples can be identical in all their attributes.***

General form :

SELECT <attribute list>
FROM <table list>
WHERE <condition>

- **<attribute list>** is a list of attribute names
- **<table list>** is a list of the relation names
- **<condition>** is a conditional (Boolean) expression

Logical Comparison Operators : =,>,<,>=,<=,<> (not equal)

The SELECT-FROM-WHERE

Example queries on company relational schema

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

The SELECT-FROM-WHERE

Example queries on company relational 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

The SELECT-FROM-WHERE : Examples

Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

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

*** the result of the query *may contain* duplicate tuples**

- **Equivalent relational algebra expression** (duplicates would not be eliminated)

$$\pi_{BDATE,\ ADDRESS}\ (\sigma_{FNAME='John'\ AND\ MINIT='B'\ AND\ LNAME='Smith'}\ (EMPLOYEE))$$

The SELECT-FROM-WHERE : Examples

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    DNUMBER=DNO  AND  DNAME='Research'
```

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

The SELECT-FROM-WHERE : Examples

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.

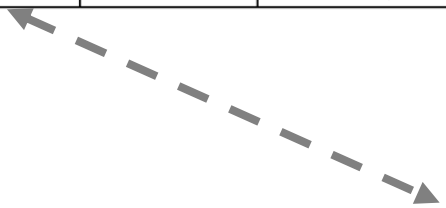
```
SELECT   PNUMBER, DNUM, LNAME, BDATE,    ADDRESS
FROM     PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER  AND MGRSSN=SSN  AND PLOCATION='Stafford'
```

In Q2, there are *two* join conditions and one select condition

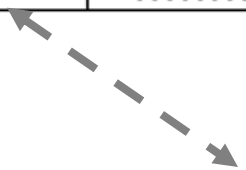
- **The join condition DNUM=DNUMBER relates a project to its controlling department.**
- **The join condition MGRSSN=SSN relates the department to the employee who manages that department.**
- **PLOCATION='Stafford' is selection condition for Project table**

The SELECT-FROM-WHERE : Examples

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



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



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

The SELECT-FROM-WHERE : Examples

Results of the SQL queries- Q0, Q1, Q2

(a)

BDATE	ADDRESS
1965-01-09	731 Fondren, 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

(c)

PNUMBER	DNUM	LNAME	ADDRESS	BDATE
10	4	Wallace	291 Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291 Berry, Bellaire, TX	1941-06-20

Ambiguous attribute names

- A query may refer to two or more attributes with the same name
- In such cases, we must *qualify* the attribute name with the relation name
- This is done by *prefixing* the relation name to the attribute name by using *dot(.)* notation to prevent ambiguity.

Ambiguous attribute names

Example:

Suppose that LNAME and DNO attributes of EMPLOYEE table were called NAME and DNUMBER, and DNAME attribute of the DEPARTMENT was also called NAME. then ***Query1 must be restated as follows.***

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

Aliasing and Tuple Variables

- Ambiguity also arises if some queries need to refer to the same relation twice
- In this case, ***aliases*** are given to the relation name using *AS*

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

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

Aliasing and Tuple Variables

- In Q8, the alternative 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***
- ***Use of AS is optional.***

  E.g. : ***Use*** **EMPLOYEE E, instead of EMPLOYEE AS E,**

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.

```
SELECT  SSN
FROM    EMPLOYEE
```

Unspecified WHERE-clause

- 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.

Q 10: Retrieve the SSN values for all employees.

```
SELECT SSN, DNAME
FROM   EMPLOYEE, DEPARTMENT;
```

Use of Asteric (*)

- Used to select all the attributes.

Q1C :

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

Q1D :

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

Q10A :

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

Q10A specifies **CROSS PRODUCT** of **EMPLOYEE** and **DEPARTMENT**

Use of DISTINCT

- **SQL does not treat a relation as a set; *duplicate tuples can appear***
- **DISTINCT is used to eliminate duplicate tuples**

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

Q11 : SELECT SALARY FROM EMPLOYEE;

Q11A : SELECT DISTINCT SALARY FROM EMPLOYEE;

(a)

SALARY
30000
40000
25000
43000
38000
25000
25000
55000

Result of Q11.

(b)

SALARY
30000
40000
25000
43000
38000
55000

Result of Q11A.

Set Theoretic Operations

- For union , the keyword is **UNION**
- For set difference the keyword is **EXCEPT**
- For set intersection the keyword is **INTERSECT**

- The relations resulting from these operations are set of tuples ; duplicates tuples are eliminated from the result.
- The relations must be ***type compatible***

Set Theoretic Operations

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

Set Theoretic Operations

Example for UNION:

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, WORKS_ON, EMPLOYEE
WHERE PNUMBER=PNO AND ESSN=SSN  AND LNAME='Smith');

   UNION

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

MultiSet Operations

3 multi set operations:

1. **UNION ALL**
2. **EXCEPT ALL**
3. **INTERSECT ALL**

- Their results are multi set (duplicates are not removed) .
- Basically, each tuple-whether it is a duplicate or not -is considered as a different tuple when applying these operations.

MultiSet Operations

Examples

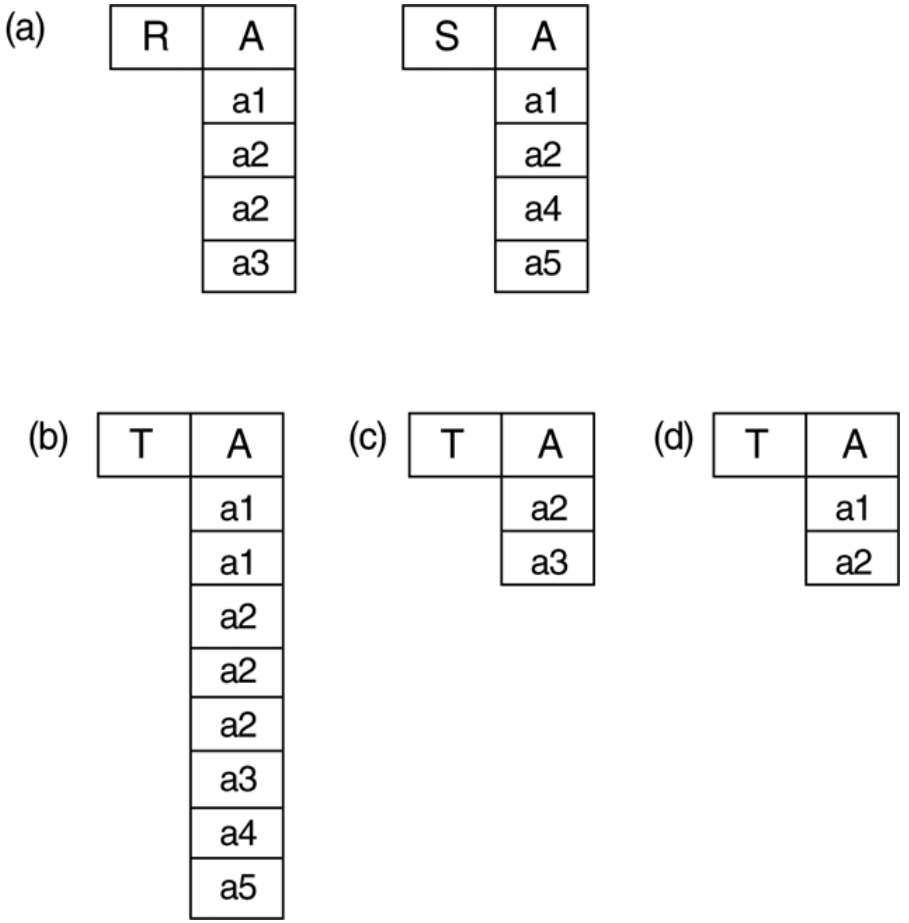


Figure 8.5 The results of SQL multiset operations. (a) Two tables, R(A) and S(A). (b) R(A) UNION ALL S(A). (c) R(A) EXCEPT ALL S(A). (d) R(A) INTERSECT ALL S(A).

Substring Pattern Matching

- **The LIKE comparison operator can be used for string pattern matching.**
- **Partial strings are specified using two reserved characters:**
  1. % replaces an arbitrary number of zero or more characters
  2. the underscore (_) replaces a single character.

QUERY 12 : Retrieve all employees whose address is in Houston.

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

Substring Pattern Matching

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

```
SELECT  FNAME, LNAME FROM EMPLOYEE
WHERE   BDATE LIKE  '195_ _ _ _ _ _ _'
```

Date format in SQL → **YYYY-MM-DD**

Substring Pattern Matching - keyword ESCAPE.

- 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

- If an apostrophe (') is needed, it is represented as two consecutive apostrophes (") so that it will not be interpreted as ending the string.

Use of Arithmetic Operators

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

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

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

Use of BETWEEN

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

Q14: **SELECT * FROM EMPLOYEE**

WHERE (SALARY BETWEEN 30000 AND 40000) AND DNO =5;

Note : The condition (SALARY BETWEEN 30000 AND 40000) in Q14 is equivalent to the condition ((SALARY >= 30000) AND (SALARY <= 40000)

Use of ORDER BY

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, 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;
```

Note : 1. The default order is in ascending order of values.

2. We can specify the keyword **DESC** if we want to see the result in a descending order of values.

More Complex SQL Queries

NULL value and Three-Valued Logic

- **SQL has 3 kinds of NULL values.**

1. **Unknown value, but existing**

   A particular person has a date of birth but it is not known.

2. **Unavailable values :**

   we do not know whether a person has a home phone number.

3. **Not applicable attribute value:**

   An attribute **CollegeDegree** would be NULL for a person who has no college degrees, because it does not apply to that person.

NULL value and Three-Valued Logic

- SQL uses a three-valued logic with values TRUE, FALSE, and UNKNOWN
- Table 8.1 shows the result of three-valued logic when logical connectives AND, OR, and NOT are used .

TABLE 8.1 LOGICAL CONNECTIVES IN THREE-VALUED LOGIC

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

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

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Use of IS or IS NOT

- SQL uses **IS** or **IS NOT** to compare an attribute value to NULL.
- This is because SQL considers each NULL value as being distinct from every other NULL value, so equality comparison is not appropriate.

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

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

Nested Queries - IN operator

- Queries can be conveniently formulated by using **nested queries**
- Nested queries are complete **select-from-where** blocks within the WHERE clause of another query (**outer query.)**
- Comparison operator **IN**, which compares a value ***v*** *with a set (multiset) of* values **V** and evaluates to **TRUE if *v is one of the elements in V***

Nested Queries - IN operator

```
SELECT  attribute_list
FROM    table_list
WHERE   attr_list_1  IN  (
                          SELECT attr_list_1
                          FROM  table_list1
                          WHERE  condition
                         )
```

Outer Query

Nested Query (inner query)

Nested Queries - IN operator

- Query 4 can be rephrased to use nested queries as shown in Q4A.

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

Q4A:

```
SELECT DISTINCT PNUMBER FROM PROJECT
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');
```

Nested Queries - IN operator

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

Nested Queries - IN operator

- The IN operator can also compare a tuple of values in parentheses with a set or multiset of union-compatible tuples.

QUERY: select the SSNs of all employees who work the same (project, hours) combination on some project that employee whose SSN ='123456789' works on.

```
SELECT DISTINCT ESSN FROM WORKS_ON
WHERE (PNO, HOURS) IN
    (SELECT PNO, HOURS FROM WORKS_ON WHERE ESSN='123456789');
```

In this example, the IN operator compares the subtuple of values in parentheses (PNO, HOURS) for each tuple in WORKS_ON is compared with the set of union-compatible tuples produced by the nested query.

Nested Queries - ANY , SOME and ALL

- Used to compare a single value v (typically an attribute name) to a set or multiset V
- **Used with =, >,>=, <, <=, and < >.**

Example : Retrieve 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 = ANY (or = SOME) operator returns TRUE if the value v is equal to some value in the set V and is hence equivalent to IN.

Nested Queries and Ambiguity

- **In general, we can have several levels of nested queries.**
- So ambiguity can arise if attributes of the same name exist in a relation in the FROM clause of the outer query, and another in a relation in the FROM clause of the nested query.
- The rule is that a reference to an unqualified attribute refers to the relation declared in the innermost nested query.

Nested Queries and Ambiguity

QUERY 16 : 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);
```

- In the nested query of *Q16*, *we must qualify* E.SEX *because it refers to the* SEX *attribute* 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 SEX of 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

Correlated Nested Queries

- **For example, we can think of Q16 as follows:**

For each EMPLOYEE tuple, evaluate the nested query, which retrieves the ESSN values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple; if the SSN value of the EMPLOYEE tuple is in the result of the nested query, then select that EMPLOYEE tuple.

Correlated Nested Queries

- In general, a query written with nested select-from-where blocks and using the = or IN comparison operators can always be expressed as a single block query. For example, Q16 may be written as in Q16A:

•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;

Correlated Nested Queries - EXISTS Function

- The **EXISTS** function in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.

- If the result of a correlated nested query contains ***at least*** one tuple , it returns TRUE . Otherwise returns FALSE

•In general, EXISTS(Q) returns TRUE if there is *at least one tuple* *in the result of the nested query Q, and it returns FALSE otherwise.*

Correlated Nested Queries - EXISTS Function

Query 16 in an alternative form that uses EXISTS. This is shown as QI6B:

```
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);
```

- **In general, EXISTS(Q) returns TRUE if there is *at least one tuple in the result of the nested query Q, and it returns FALSE otherwise.***

Correlated Nested Queries - NOT EXISTS Function

- NOT EXISTS(Q) returns TRUE if there are no tuples in the result of nested query Q, and it returns FALSE otherwise

QUERY 6 : Retrieve the names of employees who have no dependents.

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

Correlated Nested Queries - NOT EXISTS Function

We can explain Q6 as follows:

For each EMPLOYEE tuple, the correlated nested query selects all DEPENDENT tuples whose ESSN value matches the EMPLOYEE SSN; if the result is empty, no dependents are related to the employee, so we select that EMPLOYEE tuple and retrieve its FNAME and LNAME.

Correlated Nested Queries (Cont…)

QUERY 7 : List the names of managers who have at least one dependent.

```
SELECT  FNAME, LNAME FROM EMPLOYEE
WHERE
    EXISTS  (SELECT * FROM DEPENDENT WHERE SSN=ESSN)
    AND
   EXISTS  (  SELECT *  FROM DEPARTMENT WHERE    SSN=MGRSSN);
```

Correlated Nested Queries (Cont…)

QUERY 7 works as follows

1. First selects all **DEPENDENT** tuples related to an **EMPLOYEE,**
2. Second selects all **DEPARTMENT** tuples managed by the **EMPLOYEE**.
3. If at least one of the first and at least one of the second exists, we select the **EMPLOYEE** tuple.

Correlated Nested Queries (Cont...)

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

```
Q3A: SELECT FNAME, LNAME
     FROM  EMPLOYEE
     WHERE  NOT EXISTS  ((SELECT PNUMBER
                          FROM PROJECT
                          WHERE DNUM=5)

                        EXCEPT
                          (SELECT PNO  FROM WORKS_ON WHERE
                                  SSN=ESSN) );
```

Explicit Sets

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

QUERY 17 : Retrieve the social security numbers of all employees who work on project numbers 1,2, or 3.

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

Renaming of Attributes in SQL

- In SQL, it is possible to rename any attribute that appears in the result of a query by adding the qualifier **AS** followed by the desired new name. The new names will appear as column headers in the query result.

Q8A: (Retrieving the employee's name, and the name of his or her immediate supervisor.)

```
SELECT E.LNAME AS EMPLOYEE_NAME, S.LNAME AS
SUPERVISOR_NAME  FROM  EMPLOYEE AS E, EMPLOYEE AS S
WHERE  E.SUPERSSN=S.SSN;
```

Joined Tables in SQL

- Used to specify a table resulting from a join operation in the FROM clause of a query.

- **For example** consider Query1, (**Retrieving the name and address of every employee who works for the 'Research' department.)**

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

Joined Tables in SQL

- The same query can be specified **using Joined Table concept**

First specify the join of the EMPLOYEE and DEPARTMENT relations using **JOIN keyword,** and then select the desired tuples and attributes.

```
Q1A:  SELECT  FNAME, LNAME, ADDRESS
      FROM  (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
      WHERE  DNAME='Research';
```

Joined Tables in SQL

Using **NATURAL JOIN**

```
Q1B: SELECT FNAME, LNAME, ADDRESS

FROM (EMPLOYEE NATURAL JOIN

(DEPARTMENT AS DEPT (DNAME, DNO, MSSN, MSDATE)))

WHERE DNAME='Research;
```

Joined Tables in SQL

Using Multiple **JOINs**

```
Q2A:  SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
      FROM ((PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER)
            JOIN EMPLOYEE ON MGRSSN=SSN)
      WHERE  PLOCATION='Stafford';
```

Aggregate Functions in SQL-99

COUNT, SUM, MAX, MIN, and AVG functions.

- **COUNT** → returns the number of tuples or values as specified in a query.
- **SUM, MAX, MIN, and AVG** → return sum, maximum value, minimum value, and average (mean) values respectively .These are applied to a set or multiset of numeric values,
- All the functions can be used in the **SELECT** clause or in a **HAVING** clause
- The functions **MAX** and **MIN** can also be used with attributes that have nonnumeric domains

COUNT, SUM, MAX, MIN, and AVG functions.

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.

Q20: SELECT SUM (SALARY), **MAX** (SALARY), **MIN** (SALARY), **AVG** (SALARY) **FROM** EMPLOYEE ,DEPARTMENT **WHERE DNO=DNUMBER** and DNAME='Research';

COUNT, SUM, MAX, MIN, and AVG functions.

QUERY 21 : Retrieve the total number of employees in the

Q21: SELECT COUNT (*) FROM EMPLOYEE;

QUERY 22 : : Retrieve the total number of employees in the 'Research' department

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

COUNT, SUM, MAX, MIN, and AVG functions.

- We may also use the **COUNT** function to count values in a column rather than tuples, as in the next example.

QUERY 23: Count the number of distinct salary values in the database.

Q23: **SELECT COUNT** (**DISTINCT** SALARY) **FROM** EMPLOYEE;

- If we write COUNT(SALARY) instead of COUNT(DISTINCT SALARY) in Q23, then duplicate values will not be eliminated.
- Any tuples with NULL for SALARY will not be counted.
- In general, NULL values are discarded when aggregate functions are applied to a particular column (attribute).

COUNT, SUM, MAX, MIN, and AVG functions.

• We can specify a correlated nested query with an aggregate function, and then use the nested query in the WHERE clause of an outer query.

For example, to retrieve the names of all employees who have two or more dependents we can write the following:

```
SELECT  LNAME, FNAME  FROM EMPLOYEE
WHERE   (SELECT COUNT (*)
         FROM DEPENDENT
         WHERE SSN=ESSN)   >= 2
```

The GROUP BY and HAVING Clauses

The GROUP BY Clause

- The **GROUP BY** clause specifies the grouping attributes

- Grouping attributes should also appear in the SELECT clause,
- The resulting relation contains value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attributes.

The GROUP BY Clause

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

Q24: **SELECT DNO, COUNT (*), AVG (SALARY)**
FROM EMPLOYEE GROUP BY DNO;

- In 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.

The GROUP BY Clauses

QUERY 24 : Grouping EMPLOYEE tuples by the value of DNO

FNAME	MINIT	LNAME	SSN	• • •	SALARY	SUPERSSN	DNO
John	B	Smith	123456789		30000	333445555	5
Franklin		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		Jabbar	987987987		25000	987654321	4
James		Borg	888665555		55000	null	1

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

Result of Q24.

Grouping EMPLOYEE tuples by the value of DNO.

- If NULLs exist in the grouping attribute, then a separate group is created for all tuples with a *NULL value in the grouping attribute*.

The GROUP BY Clauses

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;
```

GROUP BY and HAVING Clauses

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

Q26: **SELECT** PNUMBER, PNAME, COUNT (*) **FROM** PROJECT, WORKS_ON

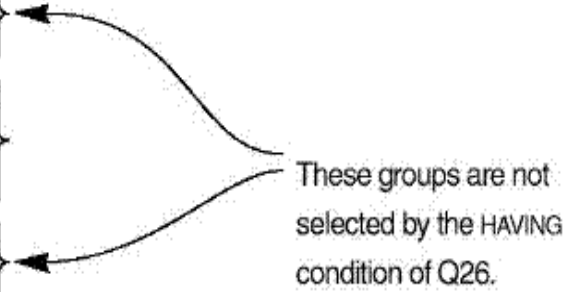
WHERE PNUMBER=PNO **GROUP BY** PNUMBER, PNAME

HAVING COUNT (*) > 2

GROUP BY and HAVING Clauses

QUERY 26 : After applying the WHERE clause but before applying HAVING

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



GROUP BY and HAVING Clauses

QUERY 26 : After applying the HAVING clause condition

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).

GROUP BY and HAVING Clauses

QUERY 27

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

```
Q27: SELECT PNUMBER, PNAME, COUNT (*)
     FROM   PROJECT, WORKS_ON, EMPLOYEE
     WHERE  PNUMBER=PNO AND SSN=ESSN AND DNO=5
     GROUP BY PNUMBER, PNAME;
```

GROUP BY and HAVING Clauses

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.

```
Q28: SELECT DNUMBER, COUNT (*)
       FROM DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO AND SALARY>40000 AND
      DNO IN    (SELECT DNO  FROM EMPLOYEE
                   GROUP BY DNO
                   HAVING COUNT (*) > 5)
GROUP BY DNUMBER;
```

INSERT, DELETE, AND UPDATE STATEMENTS IN SQL-99

The INSERT Command

- **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.

Example : Adding a new tuple to employee table

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

Another form of the INSERT Command :

- This allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command.
- This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.
- However, the values must include all attributes with NOT NULL specification and no default value.
- Attributes with NULL allowed or DEFAULT values are the ones that can be left out.

Another form of the INSERT Command :

For example, to enter a tuple for a new EMPLOYEE for whom we know only the FNAME, LNAME, DNO, and SSN attributes, we can use U1A:

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

- Attributes not specified in U lA are set to their DEFAULT or to NULL

Inserting multiple tuples using select

• A variation of the INSERT command inserts multiple tuples into a relation in conjunction with creating the relation and loading it with the result of a query.

• **For example,** to create a temporary table that has the name, number of employees, and total salaries for each department, we can write the statements in U3A and U3B:

U3A: **CREATE TABLE DEPTS_INFO** (

DEPT_NAME **VARCHAR(15),**

NO_OF_EMPS **INTEGER,**

TOTAL_SAL **INTEGER**

);

Inserting multiple tuples using select

U3B: **INSERT INTO DEPTS_INFO** (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)

SELECT DNAME, **COUNT (*), SUM (SALARY)**

FROM DEPARTMENT , **EMPLOYEE ON**

WHERE DNUMBER=DNO

GROUP BY DNAME;

• A table DEPTS_INFO is created by U3A and is loaded with the summary information retrieved from the database by the query in U3B.

The DELETE Command

- The DELETE command removes tuples from a relation. It can include WHERE clause to select the tuples to be deleted.
- Tuples are explicitly deleted from only one table at a time
- The deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential integrity constraints of the DDL
- A missing WHERE clause specifies that all tuples in the relation are to be deleted;

The DELETE Command

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; (becomes empty table)

The UPDATE Command

- The UPDATE command is used to modify attribute values of one or more selected tuples.

- As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation.

- However, updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL

- An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values.

The UPDATE Command

- For example, to change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively, we use US:

- **U5:** **UPDATE** PROJECT **SET** PNUMBER=10 , DNUM = 5 **WHERE** PLOCATION = 'Bellaire‘;

The UPDATE Command

- Several tuples can be modified with a single UPDATE command. An example is to give all employees in the 'Research' department a 10 percent raise in salary, as shown in U6.

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

- It is also possible to specify NULL or DEFAULT as the new attribute value.
- Notice that each UPDATE command explicitly refers to a single relation only.

VIEWS (VIRTUAL TABLES) IN SQL

Concept of a View in SQL

- A view is a single table that is derived from other tables
- other tables → **base tables** or **previously defined views**
- A view does not necessarily exist in **physical form.**
- Views are useful when a table needs to be reference frequently

Why a View in SQL?

- Suppose that we frequently issue queries that retrieve the **employee name and the project names that the employee works on.**

```
SELECT   FNAME, LNAME, PNAME, HOURS
FROM     EMPLOYEE, PROJECT, WORKS_ON
WHERE    SSN=ESSN AND PNO=PNUMBER;
```

Rather than having to specify the join of the EMPLOYEE, WORKS_ON, and PROJECT tables every time we issue that query , we can define a view that is a result of these joins. We can then issue queries on the view

Specification of Views in SQL

- the command to specify a view is **CREATE VIEW.**
- The view is given a
    1. **view name,**
    2. **a list of attribute names (Optional)**
    3. **and a query to specify the contents of the view.**

```
V1: CREATE VIEW    WORKS_ON1
    AS  SELECT     FNAME, LNAME, PNAME, HOURS
        FROM       EMPLOYEE, PROJECT, WORKS_ON
        WHERE      SSN=ESSN AND PNO=PNUMBER;
```

Specification of Views in SQL

```
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

Specification of Views in SQL

We can now specify SQL queries on a view-or virtual table-in the same way we specify queries involving base tables.

Example : to retrieve the last name and first name Of all employees who work on 'ProjectX',

```
QV1: SELECT  FNAME, LNAME
     FROM    WORKS_ON1
     WHERE   PNAME='ProjectX';
```

Equivalent query if if specified on the base relations

```
SELECT  FNAME, LNAME
FROM    EMPLOYEE, PROJECT, WORKS_ON
WHERE   SSN=ESSN AND PNO=PNUMBER
        AND PNAME='ProjectX';
```

Specification of Views in SQL

Advantages

- a view is to simplify the specification of certain queries.
- Views are also used as a security and authorization mechanism

- ***A view is supposed to be always up to date;***

- DROP VIEW command is used to dispose a view

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