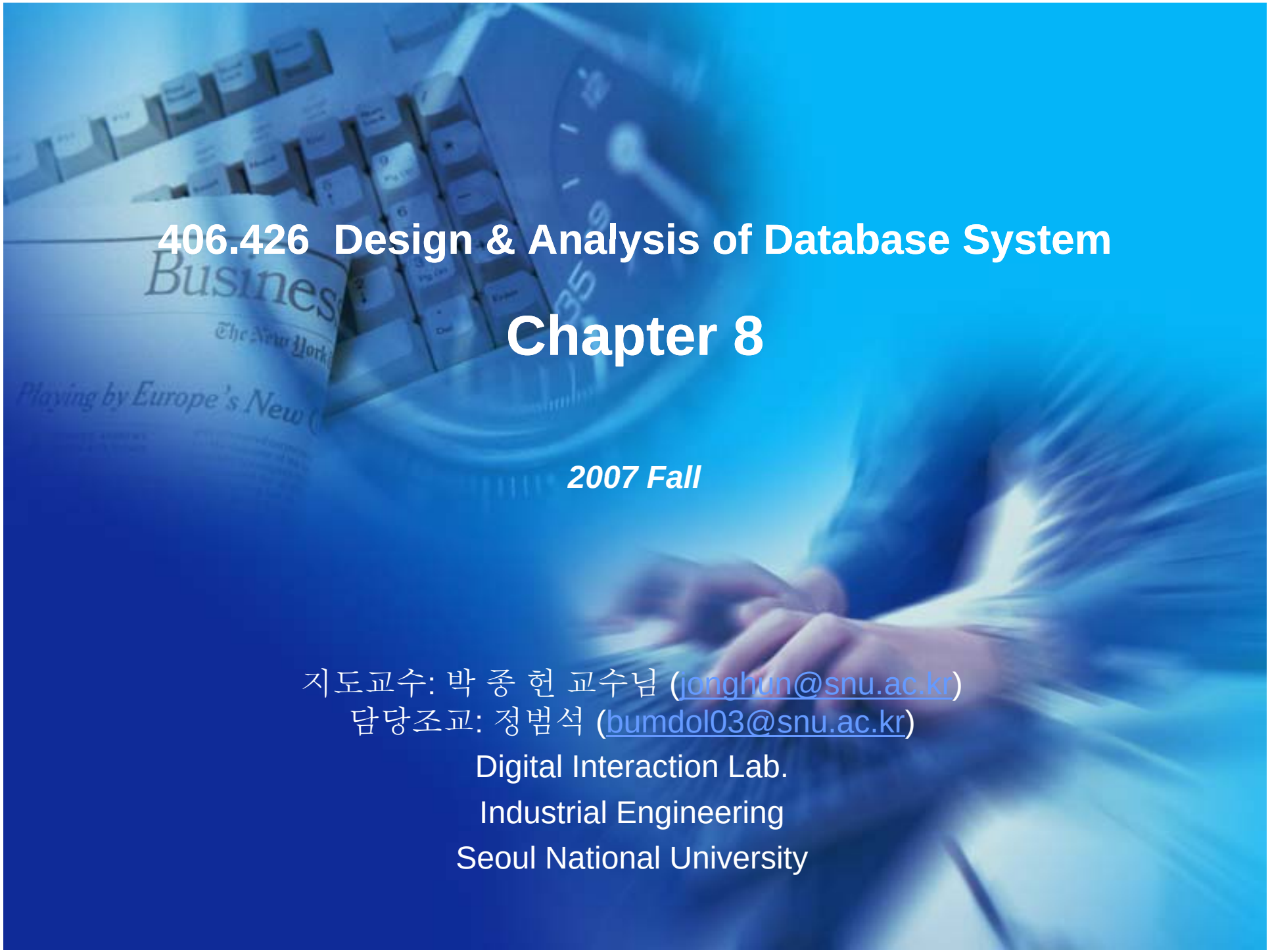


406.426 Design & Analysis of Database System

Chapter 8

2007 Fall

지도교수: 박 종 헌 교수님 (jonghun@snu.ac.kr)
담당조교: 정범석 (bumdol03@snu.ac.kr)
Digital Interaction Lab.
Industrial Engineering
Seoul National University

Create Table

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types

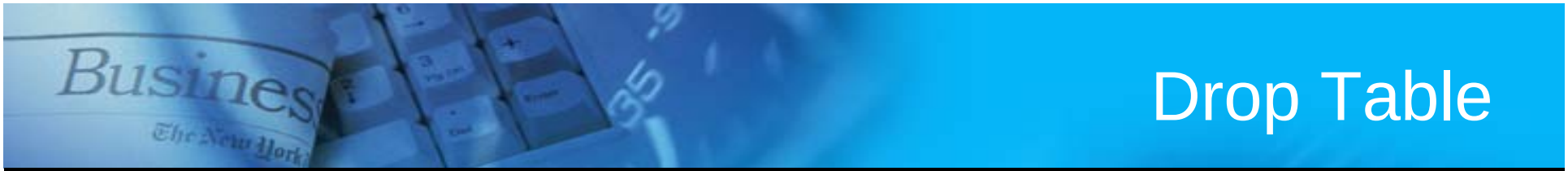
- A constraint NOT NULL may be specified on an attribute

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

Create Table

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

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



Drop Table

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

DROP TABLE DEPENDENT;

Alter Table

- 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.

Referential Integrity Constraint

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

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

Additional Data Types in SQL2 and SQL99

- **DATE:**
  - Made up of year-month-day in the format yyyy-mm-dd
- **TIME:**
  - Made up of hour:minute:second in the format hh:mm:ss
- **TIME(i):**
  - Made up of hour:minute:second plus i additional digits specifying fractions of a second
  - format is hh:mm:ss:ii...i
- **TIMESTAMP:**
  - Has both DATE and TIME components
- **INTERVAL:**
  - Specifies a relative value rather than an absolute value
  - Can be DAY/TIME intervals or YEAR/MONTH intervals
  - Can be positive or negative when added to or subtracted from an absolute value, the result is an absolute value

Create Table Example

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

Create Table Example (cont.)

```
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) ) ;
```

Create Table with Constraints

```
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) ) ;
```

Create Table with Constraints (cont.)

```
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 );

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 ) ;
```

Create Table with Constraints (cont.)

```
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 );

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 ) ;
```

Retrieval Queries in SQL

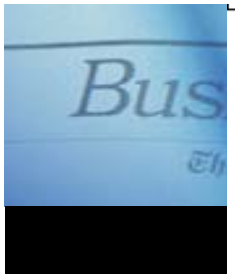
- SQL has one basic statement for retrieving information from a database; the SELECT statement
- This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model; SQL allows a table (relation) to have two or more tuples that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

Retrieval Queries in SQL

- 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



John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS

DNUMBER	DLOCATION
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

DEPARTMENT

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

WORKS_ON

ESSN	PNO	HOURS
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	null

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

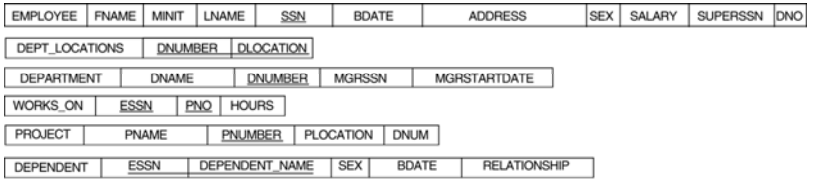
ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
333445555	Alice	F	1986-04-05	DAUGHTER
333445555	Theodore	M	1983-10-25	SON
333445555	Joy	F	1958-05-03	SPOUSE
987654321	Abner	M	1942-02-28	SPOUSE
123456789	Michael	M	1988-01-04	SON
123456789	Alice	F	1988-12-30	DAUGHTER
123456789	Elizabeth	F	1967-05-05	SPOUSE

Simple Query

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

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

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

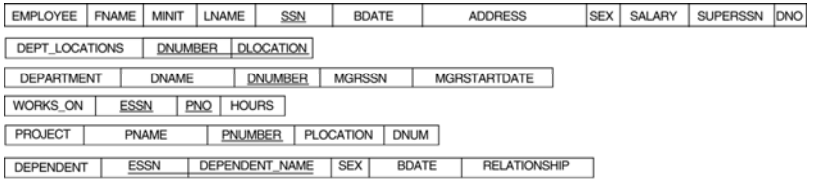


Simple Query (cont.)

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

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

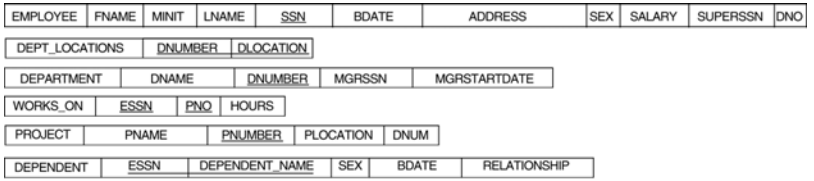
- Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
- (DNAME='Research') is a *selection condition* (corresponds to a SELECT operation in relational algebra)
- (DNUMBER=DNO) is a *join condition* (corresponds to a JOIN operation in relational algebra)



Alias

❖ 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*

- **Q8:**

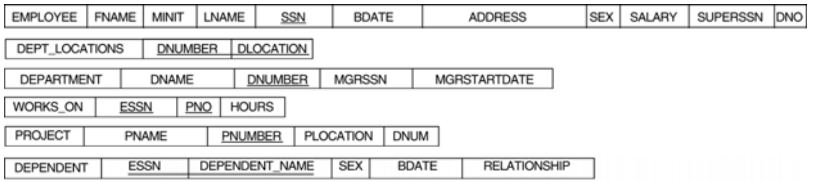
```
SELECT  E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM    EMPLOYEE AS E, EMPLOYEE AS S
WHERE   E.SUPERSSN=S.SSN
```

Unspecified Where-clause

- A *missing WHERE-clause* indicates no condition; hence, *all tuples* of the relations in the FROM-clause are selected

- Query 9: Retrieve the SSN values for all employees.

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



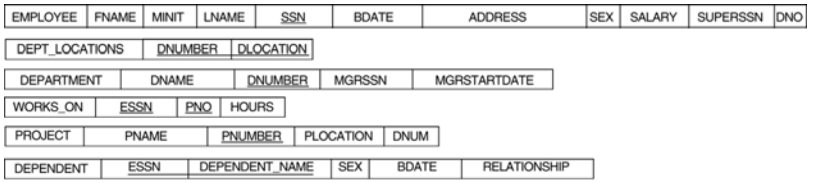
- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected

- **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:

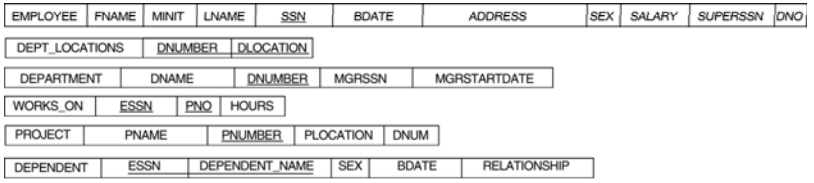
Q1C:
```
SELECT      *
FROM  EMPLOYEE
WHERE       DNO=5
```

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
```

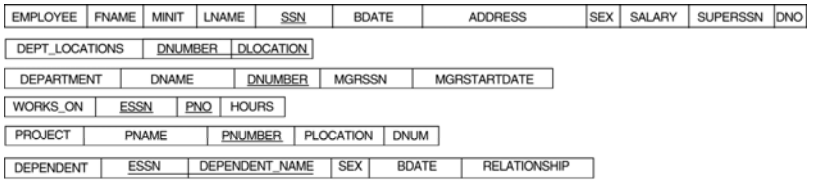


Set Operation

- 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

- 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.



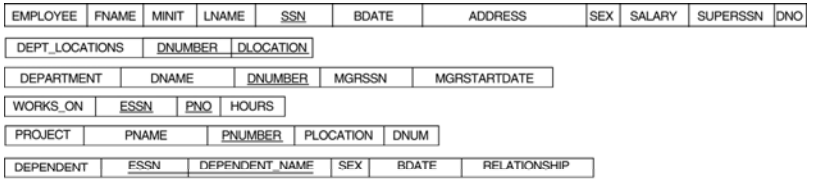
-

```
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')
```

Nesting of Query

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*

- 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' )
```

- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query

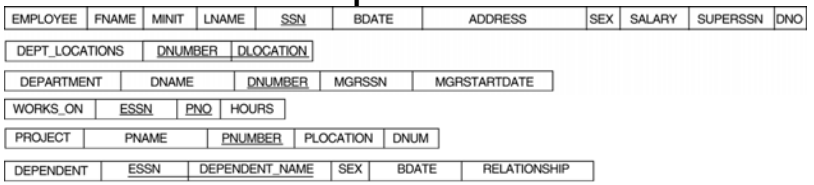
- In this example, the nested query is *not correlated* with the outer query

Correlated Nested Query

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated*

- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*

- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

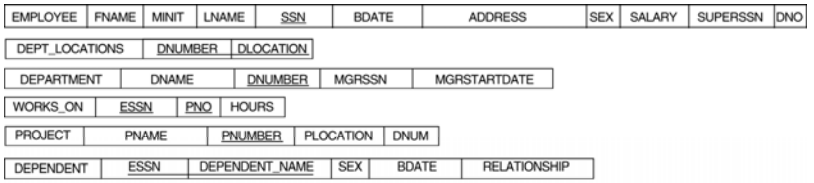



```
Q12: SELECT      E.FNAME, E.LNAME
     FROM        EMPLOYEE AS E
     WHERE E.SSN IN (SELECT      ESSN
                     FROM    DEPENDENT
                     WHERE ESSN=E.SSN AND
                           E.FNAME=DEPENDENT_NAME)
```

Correlated Nested Query (cont.)

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

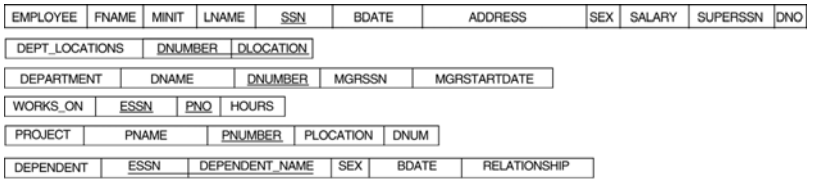
```
Q12A:   SELECT   E.FNAME, E.LNAME
        FROM     EMPLOYEE E, DEPENDENT D
        WHERE          E.SSN=D.ESSN AND
                 E.FNAME=D.DEPENDENT_NAME
```



Correlated Nested Query (cont.)

- The CONTAINS operator compares two *sets of values* , and returns TRUE if one set contains all values in the other set

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




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

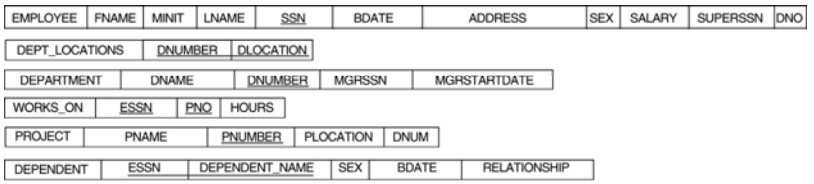
- In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5

- The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different *for each employee tuple* because of the correlation

Exist Function

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

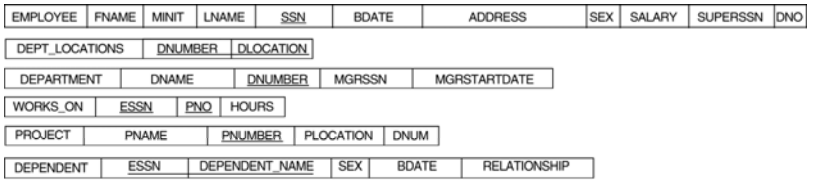
```
Q12B:   SELECT      FNAME, LNAME
        FROM        EMPLOYEE
        WHERE       EXISTS
        (SELECT * FROM      DEPENDENT
                WHERE SSN=ESSN AND
                FNAME=DEPENDENT_NAME)3
```



Exist Function (cont.)

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

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



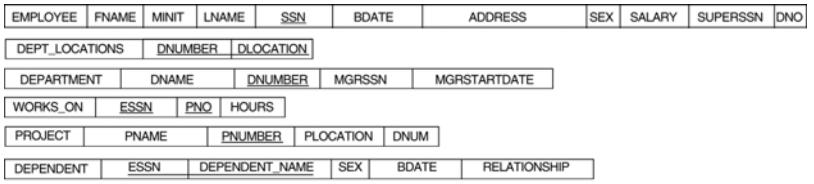
  - In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected

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)
```



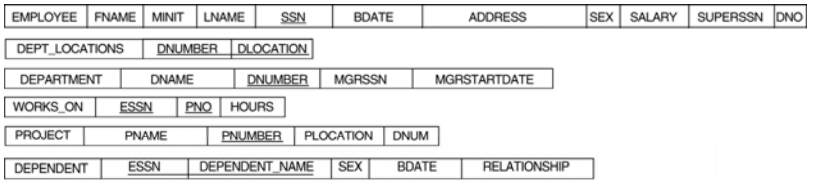
Nulls in SQL Queries

- 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

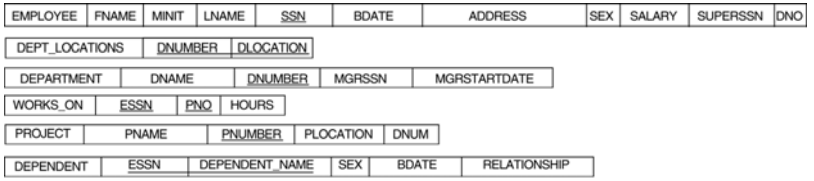
Joined Relation

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join
- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

Joined Relation (cont.)

- Examples:

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



can be written as:

Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM (EMPLOYEE E LEFT OUTER JOIN EMPLOYEES
ON E.SUPERSSN=S.SSN)

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

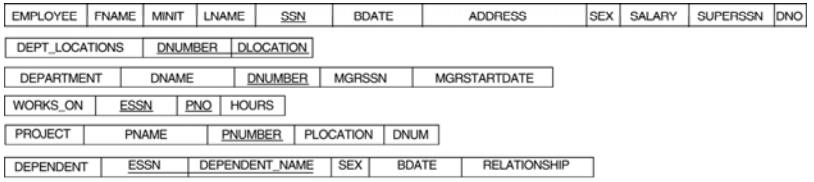
Aggregate Functions

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**

- **Q15: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)**
  FROM EMPLOYEE

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

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

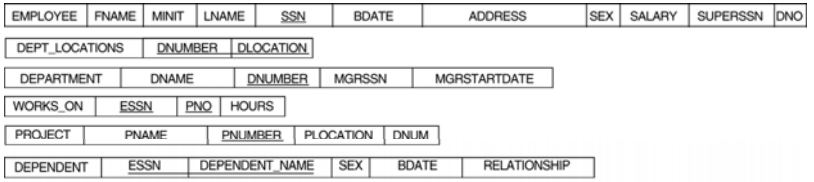


- **Q18: SELECT COUNT (*)**
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNO=DNUMBER AND
  DNAME='Research'

Grouping

- **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
```



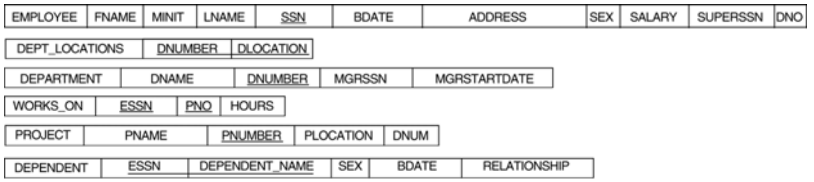
- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

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

Having Clouse

- 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
```



Result of Group by and Having

(a)

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	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.

Result of Group by and Having (cont.)

(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.

After applying the WHERE clause but before applying HAVING.

Result of Group by and Having (cont.)

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

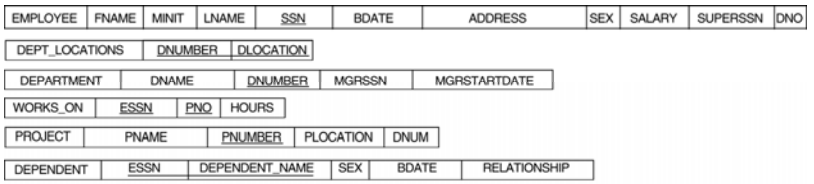
After applying the HAVING clause condition.

PNAME	COUNT (*)
ProductY	
Computerization	
Reorganization	
Newbenefits	

Result of Q26
(PNUMBER not shown).

Substring Comparison

- 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%'
```

- **Q26:SELECT FNAME, LNAME**
  FROM EMPLOYEE
  WHERE BDATE LIKE '_____5_'

Arithmetic Operations

- The standard arithmetic operators '+', '-'. '*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result

- Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

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

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPT_LOCATIONS	DNUMBER	DLOCATION

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

WORKS_ON	ESSN	PNO	HOURS

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Order by

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

- Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPT_LOCATIONS	DNUMBER	DLOCATION

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

WORKS_ON	ESSN	PNO	HOURS

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

-

```
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
```

Order by

- 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

Insert Statement

- 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 Statement

- 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 Statement

- 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 ;
```

Delete Statement

- 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 Statement

- 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 Statement

- 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 Statement

- 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 Statement

- 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

Assertions

- General constraints: constraints that do not fit in the basic SQL categories (presented in chapter 8)
- Mechanism: `CREAT ASSERTION`
  - components include: a constraint name, followed by `CHECK`, followed by a condition
- Specify a query that violates the condition; include inside a `NOT EXISTS` clause
- Query result must be empty
  - if the query result is not empty, the assertion has been violated

Assertions

- "The salary of an employee must not be greater than the salary of the manager of the department that the employee works for"

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT *
 FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
 WHERE E.SALARY > M.SALARY AND
       E.DNO=D.NUMBER AND D.MGRSSN=M.SSN))
```

Views

- SQL command: `CREATE VIEW`
  - a table (view) name
  - a possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations)
  - a query to specify the table contents

```
CREATE TABLE WORKS_ON_NEW AS
   SELECT FNAME, LNAME, PNAME, HOURS
       FROM EMPLOYEE, PROJECT, WORKS_ON
   WHERE SSN=ESSN AND PNO=PNUMBER
       GROUP BY PNAME;
```

Virtual Table

- We can specify SQL queries on a newly create table (view):

```
SELECT FNAME, LNAME FROM WORKS_ON_NEW
WHERE PNAME='Seena';
```

- When no longer needed, a view can be dropped:

```
DROP WORKS_ON_NEW;
```

Efficient View Implementation

- Query modification: present the view query in terms of a query on the underlying base tables
  - disadvantage: inefficient for views defined via complex queries (especially if additional queries are to be applied to the view within a short time period)

- View materialization: involves physically creating and keeping a temporary table
  - assumption: other queries on the view will follow
  - concerns: maintaining correspondence between the base table and the view when the base table is updated
  - strategy: incremental update

View Update

- Update on a single view without aggregate operations: update may map to an update on the underlying base table

- Views involving joins: an update *may* map to an update on the underlying base relations
  - not always possible

Un-updatable Views

- Views defined using groups and aggregate functions are not updateable
- Views defined on multiple tables using joins are generally not updateable
- `WITH CHECK OPTION`: must be added to the definition of a view if the view is to be updated
  - to allow check for updatability and to plan for an execution strategy


Thank You !