SQL: QUERIES 2

Week 12

Relational Database 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

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

SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a union operation (**UNION)**, and in *some versions* of SQL there are set difference (**MINUS)** and intersection (**INTERSECT)** operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations* ; the two relations must have the same attributes and the attributes must appear in the same order

SET OPERATIONS (cont.)

- ***Query 9:*** ***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.***

Q9:

```
(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 QUERIES

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

- Many of the previous queries can be specified in an alternative form using nesting

- ***Query 10:*** ***Retrieve the name and address of all employees who work for the 'Research' department.***

Q10:
```
SELECT FNAME, LNAME, ADDRESS
FROM   EMPLOYEE
WHERE  DNO IN  (SELECT  DNUMBER
FROM   DEPARTMENT
WHERE  DNAME='Research' )
```

NESTING OF QUERIES (cont.)

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator **IN** compares a value v with a set (or multi-set) of values V, and evaluates to **TRUE** if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In this example, the nested query is *not correlated* with the outer query

CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query*, the two queries are said to be *correlated*
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*

- ***Query 11:* Retrieve the name of each employee who has a dependent with the same first name as the employee.**

Q11:
```
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 QUERIES (cont.)

- In Q11, the nested query has a different result *for each tuple* in the outer query

- 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, Q11 may be written as in Q11A

Q11A:
```
SELECT       E.FNAME, E.LNAME
FROM   EMPLOYEE E, DEPENDENT D
WHERE  E.SSN=D.ESSN AND E.FNAME=D.DEPENDENT_NAME
```

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query

- ***Query 14:*** ***Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.***

Q14:

```
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 15:* Retrieve the names of all employees who do not have supervisors.**

Q15:
```
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

AGGREGATE FUNCTIONS

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

- ***Query 17:*** ***Find the maximum salary, the minimum salary, and the average salary among all employees.***

Q17:

```
SELECT  MAX(SALARY),
        MIN(SALARY), AVG(SALARY)
FROM  EMPLOYEE
```

  - Some SQL implementations *may not allow more than one function* in the SELECT-clause

AGGREGATE FUNCTIONS (cont.)

- Query 18: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

Q18:
```
SELECT MAX(SALARY), MIN(SALARY),
       AVG(SALARY)
FROM  EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```

AGGREGATE FUNCTIONS (cont.)

- ***Queries 19 and 20:*** ***Retrieve the total number of employees in the company (Q19), and the number of employees in the 'Research' department (Q20).***

Q19:

```
SELECT     COUNT (*)
FROM  EMPLOYEE
```

Q20:

```
SELECT  COUNT (*)
FROM  EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research’
```

SUBSTRING COMPARISON

- The **LIKE** comparison operator is used to compare partial strings

- Two reserved characters are used: '**%**' (or '*' in some implementations) **replaces an arbitrary number of characters**,
- and '_' **replaces a single arbitrary character**

SUBSTRING COMPARISON (cont.)

- ***Query 24:*** ***Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.***

Q24:

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

SUBSTRING COMPARISON (cont.)

- ***Query 25:*** ***Retrieve all employees who were born during the 1950s.*** Here, '5' must be the 9th character of the string (e.g. 18-11-1950), so the BDATE value is '_____**5**_', with each underscore as **a place holder for a single arbitrary character**.

  Q25:

```
SELECT    FNAME, LNAME
FROM  EMPLOYEE
WHERE BDATE LIKE '_____5_'
```

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible; hence, in SQL, character string attribute values are not atomic

ARITHMETIC OPERATIONS

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

- ***Query 26:*** ***Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.***

```
Q26: SELECT FNAME, LNAME, 1.1*SALARY
     FROM  EMPLOYEE, WORKS_ON, PROJECT
     WHERE SSN=ESSN AND PNO=PNUMBER AND
           PNAME='ProductX’
```

Joined Relations

- 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 Relations (cont.)

- Examples:

Q15:
```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM   EMPLOYEE E S
WHERE  E.SUPERSSN=S.SSN
```

can be written as:

Q15A:
```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM (EMPLOYEE E LEFT OUTER JOIN EMPLOYEE S
        ON  E.SUPERSSN=S.SSN)
```

Joined Relations (cont.)

- **Q16:**

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

- could be written as:

Q16A:

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

Do not use **ON** clause

or as:

Q16B:

```
SELECT FNAME, LNAME, ADDRESS
FROM   (EMPLOYEE NATURAL JOIN DEPARTMENT
WHERE  DNAME='Research’
```

GROUPING

- In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation*
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

GROUPING (cont.)

- ***Query 21:*** ***For each department, retrieve the department number, the number of employees in the department, and their average salary.***

  Q21:
  ```
  SELECT DNO, COUNT (*), AVG (SALARY)
  FROM   EMPLOYEE
  GROUP BY DNO
  ```

  - In Q21, the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO
  - The COUNT and AVG functions are applied to each such group of tuples separately
  - The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
  - A join condition can be used in conjunction with grouping

GROUPING (cont.)

- ***Query 22:*** ***For each project, 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
```

  - In this case, the grouping and functions are applied *after* the joining of the two relations

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*

- The HAVING-clause is used for **specifying a selection condition on groups** (rather than on individual tuples)

THE HAVING-CLAUSE (cont.)

- ***Query 23:*** ***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.***

Q23:

```
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT (*) > 2
```

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 27:*** ***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.***

```
Q27: 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 (cont.)

- 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 With Select

- To copy all columns from one table to another, existing table

```
INSERT INTO table2
SELECT * FROM table1;
```

INSERT WITH SELECT

- Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table DEPTS_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

U3A:

```
CREATE TABLE  DEPTS_INFO
       (DEPT_NAME   VARCHAR(10),
        NO_OF_EMPS  INTEGER,
        TOTAL_SAL   INTEGER);
```

U3B:

```
INSERT INTO  DEPTS_INFO (DEPT_NAME,
       NO_OF_EMPS, TOTAL_SAL)
SELECT       DNAME, COUNT (*), SUM (SALARY)
FROM         DEPARTMENT, EMPLOYEE
WHERE        DNUMBER=DNO
GROUP BY     DNAME ;
```

INSERT WITH SELECT

- Note: The DEPTS_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations *after* issuing U3B.

- We have to create a **view** to keep such a table up to date.

DELETE WITH SELECT

- Examples:

- **Delete employees in Research department**

U4C:

```
DELETE FROM     EMPLOYEE
WHERE       DNO   IN
(SELECT     DNUMBER
      FROM  DEPARTMENT
      WHERE DNAME='Research‘)
```

UPDATE WITH SELECT

- ***Example:* Give all employees in the 'Research' department a 10% raise in salary.**

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

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
- The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
- The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

```
SELECT <attribute list>
FROM <table list>
[WHERE    <condition>]
[GROUP BY <grouping attribute(s)>]
[HAVING   <group condition>]
[ORDER BY <attribute list>]
```

Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause