ENSF 608: SQL

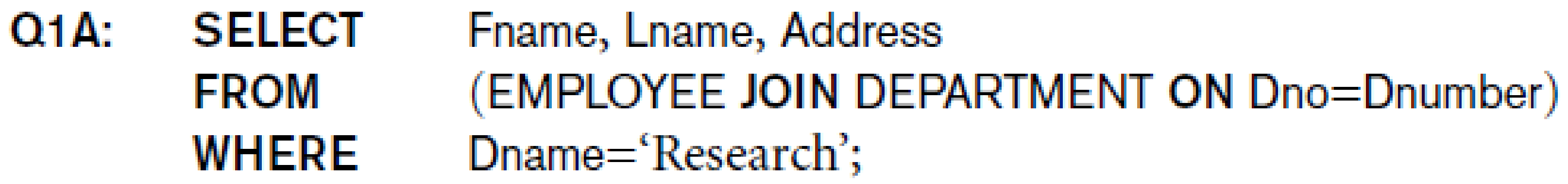
Dr. Emily Marasco

Fall 2021

Textbook: Fundamentals of Database Systems, 7th Ed., Elmasri & Navathe

Specifying Joined Tables in the FROM Clause of SQL

- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query
- The FROM clause in Q1A
  - Contains a single joined table. JOIN may also be called INNER JOIN



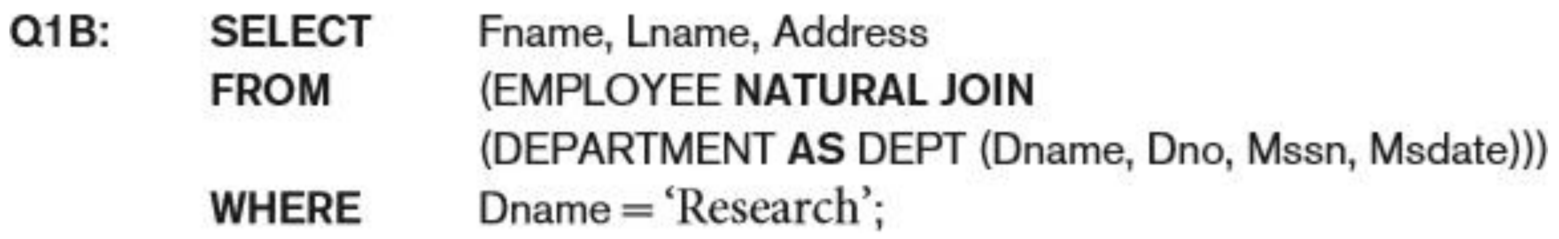
```
Q1A:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
        WHERE     Dname='Research';
```

Different Types of JOINed Tables in SQL

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN (LEFT, RIGHT, FULL)
- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S

NATURAL JOIN

- Rename attributes of one relation so it can be joined with another using NATURAL JOIN:



```
Q1B:    SELECT   Fname, Lname, Address
        FROM     (EMPLOYEE NATURAL JOIN
                 (DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
        WHERE    Dname = 'Research';
```

The above works with EMPLOYEE.Dno = DEPT.Dno as an implicit join condition

INNER and OUTER Joins

- INNER JOIN (**versus** OUTER JOIN)
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation
- LEFT OUTER JOIN
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table
- RIGHT OUTER JOIN
  - Every tuple in right table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of left table

Example: LEFT OUTER JOIN

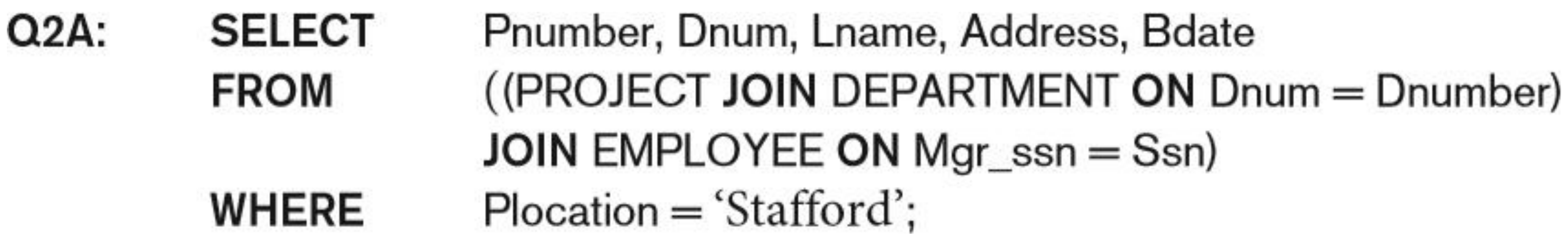
```
SELECT      E.Lname AS Employee_name,
            S.Lname AS Supervisor_name
FROM        (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S
            ON E.Super_ssn = S.Ssn);
```

Alternate Syntax:

```
SELECT      E.Lname, S.Lname
FROM        EMPLOYEE E, EMPLOYEE S
WHERE       E.Super_ssn + = S.Ssn;
```

Multiway JOIN in the FROM Clause

- Can nest JOIN specifications for a multiway join:



```
Q2A:    SELECT    Pnumber, Dnum, Lname, Address, Bdate
        FROM      ((PROJECT JOIN DEPARTMENT ON Dnum = Dnumber)
                  JOIN EMPLOYEE ON Mgr_ssn = Ssn)
        WHERE     Plocation = 'Stafford';
```

Aggregate Functions in SQL (1 of 3)

- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
  - **`COUNT`**, **`SUM`**, **`MAX`**, **`MIN`**, and **`AVG`**
- **Grouping**
  - Create subgroups of tuples before summarizing
- To select entire groups, `HAVING` clause is used
- Aggregate functions can be used in the `SELECT` clause or in a `HAVING` clause

Renaming Results of Aggregation

- Following query returns a single row of computed values from EMPLOYEE table:

```
Q19:    SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
        FROM      EMPLOYEE;
```

- The result can be presented with new names:

```
Q19A:   SELECT    SUM (Salary) AS Total_Sal, MAX (Salary) AS Highest_Sal,
                  MIN (Salary) AS Lowest_Sal, AVG (Salary) AS Average_Sal
        FROM      EMPLOYEE;
```

Aggregate Functions in SQL (2 of 3)

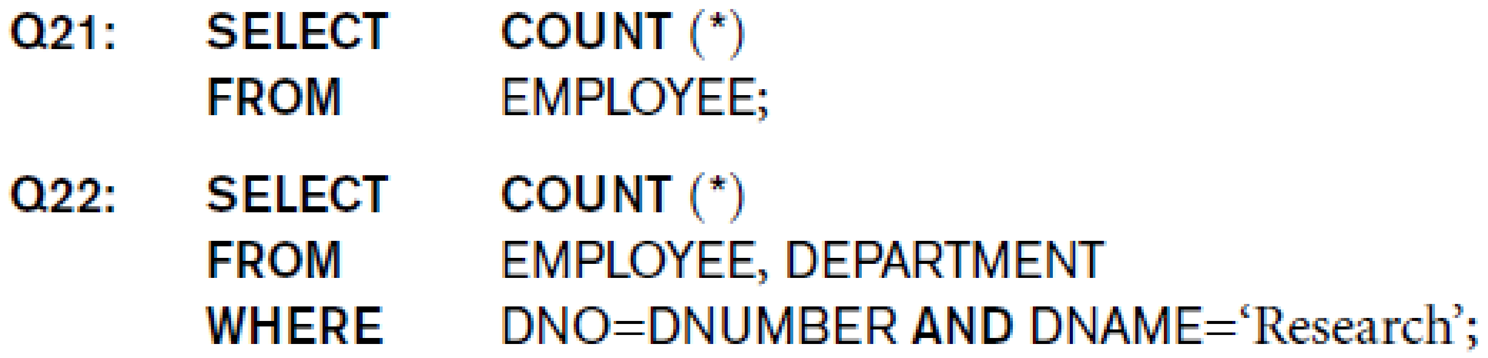
- NULL values are discarded when aggregate functions are applied to a particular column

- **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 JOIN DEPARTMENT ON Dno=Dnumber)
        WHERE     Dname=‘Research’;
```

Aggregate Functions in SQL (3 of 3)

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



```
Q21:    SELECT    COUNT (*)
        FROM      EMPLOYEE;

Q22:    SELECT    COUNT (*)
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DNO=DNUMBER AND DNAME='Research';
```

Grouping: The GROUP BY Clause

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **`GROUP BY`** clause
  - Specifies grouping attributes
- COUNT (*) counts the number of rows in the group

Examples of GROUP BY

- The grouping attribute must appear in the SELECT clause:

```
Q24:    SELECT      Dno, COUNT (*), AVG (Salary)
        FROM        EMPLOYEE
        GROUP BY    Dno;
```

- If the grouping attribute has NULL as a possible value, then a separate group is created for the null value (e.g., null Dno in the above query)
- GROUP BY may be applied to the result of a JOIN:

```
Q25:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber = Pno
        GROUP BY    Pnumber, Pname;
```

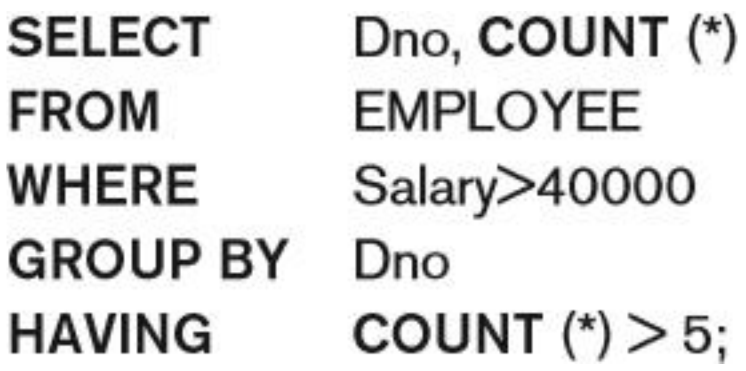
Grouping: The GROUP BY and HAVING Clauses

- **`HAVING`** clause
  - Provides a condition to select or reject an entire group:
- **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;
```

Combining the WHERE and the HAVING Clause (1 of 2)

- WHERE is applied first, then the HAVING clause
- Consider the query: we want to count the **total** number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.
- Incorrect Query:



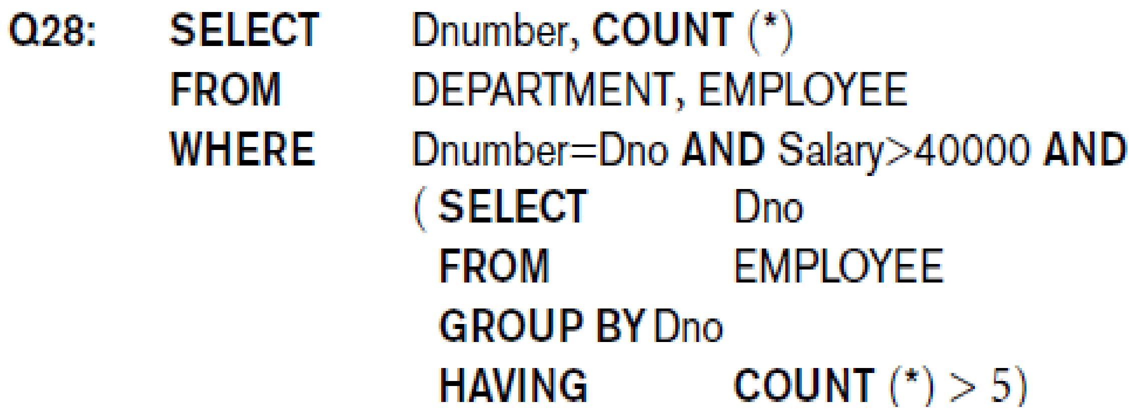
```
SELECT      Dno, COUNT (*)
FROM        EMPLOYEE
WHERE       Salary>40000
GROUP BY    Dno
HAVING      COUNT (*) > 5;
```

Combining the WHERE and the HAVING Clause (2 of 2)

Correct Specification of the Query:

- Note: the WHERE clause applies tuple by tuple whereas HAVING applies to entire group of tuples

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.



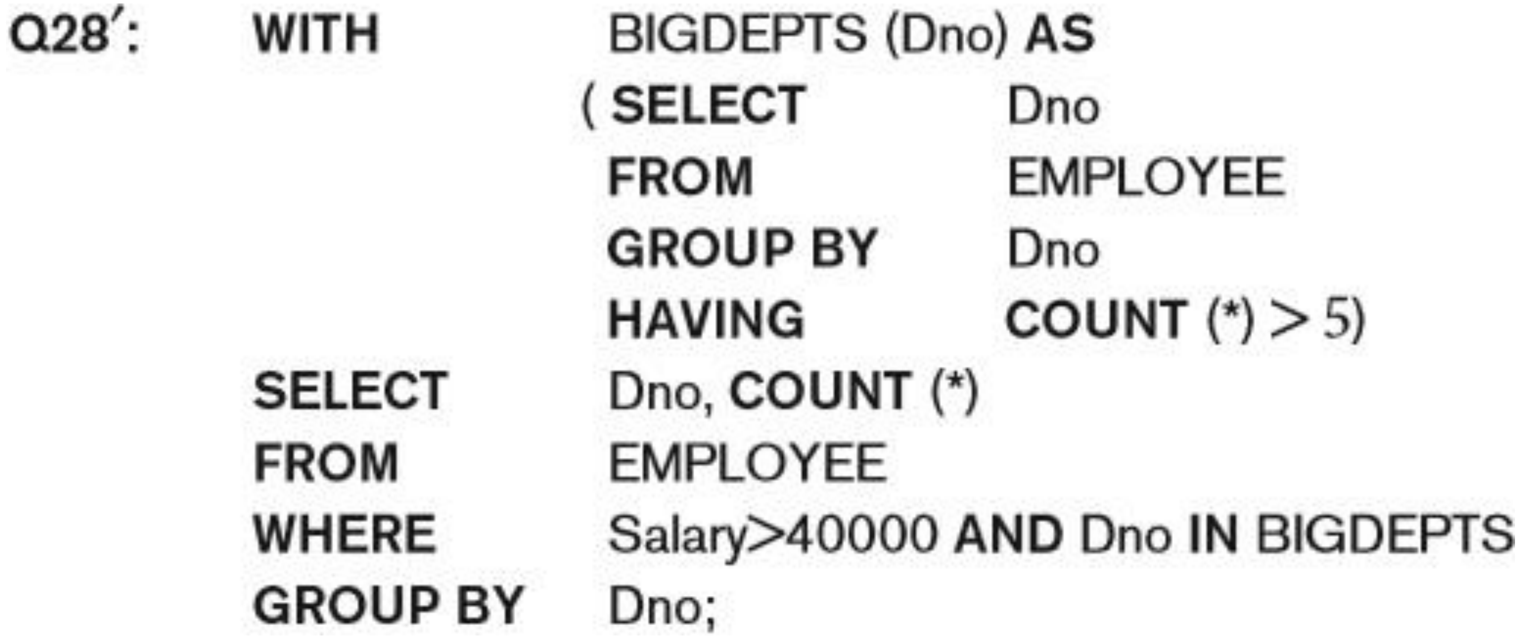
```
Q28:    SELECT      Dnumber, COUNT (*)
        FROM        DEPARTMENT, EMPLOYEE
        WHERE       Dnumber=Dno AND Salary>40000 AND
                    ( SELECT         Dno
                      FROM           EMPLOYEE
                      GROUP BY Dno
                      HAVING         COUNT (*) > 5)
```

Use of WITH

- The WITH clause allows a user to define a table that will only be used in a particular query (not available in all SQL implementations)
- Used for convenience to create a temporary “View” and use that immediately in a query
- Allows a more straightforward way of looking a step-by-step query

Example of WITH

- See an alternate approach to doing Q28:



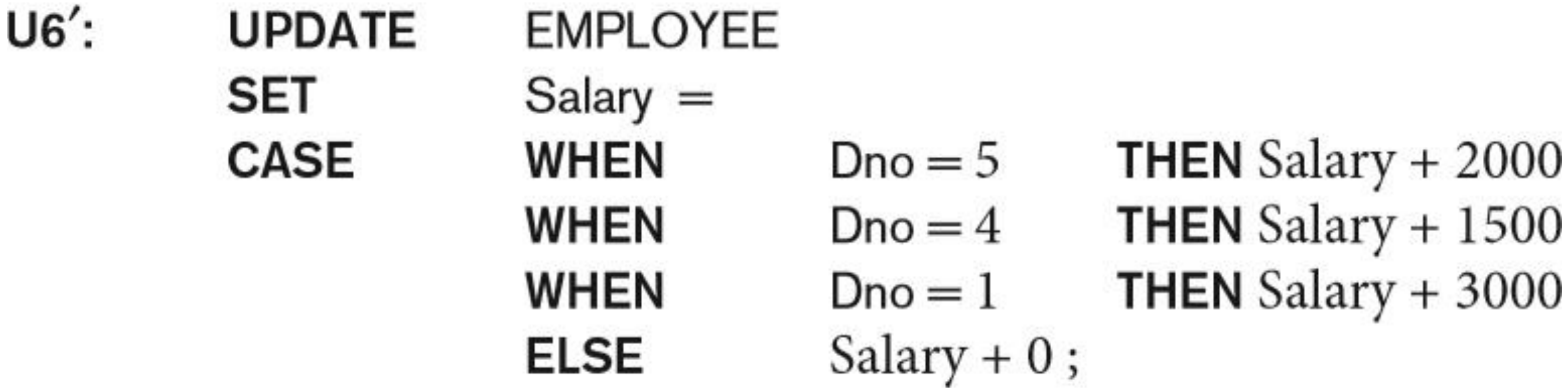
```
Q28′:   WITH       BIGDEPTS (Dno) AS
                   ( SELECT      Dno
                     FROM        EMPLOYEE
                     GROUP BY    Dno
                     HAVING      COUNT (*) > 5)
        SELECT     Dno, COUNT (*)
        FROM       EMPLOYEE
        WHERE      Salary>40000 AND Dno IN BIGDEPTS
        GROUP BY   Dno;
```

Use of CASE

- SQL also has a CASE construct
- Used when a value can be different based on certain conditions.
- Can be used in any part of an SQL query where a value is expected
- Applicable when querying, inserting or updating tuples

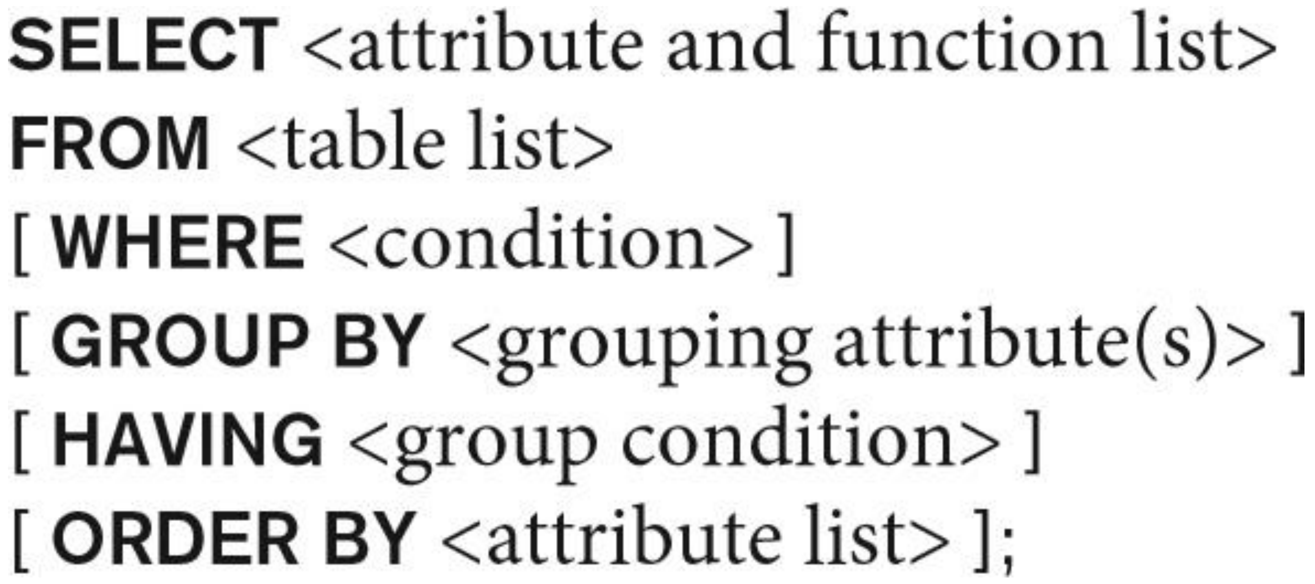
EXAMPLE of Use of CASE

- The following example shows that employees are receiving different raises in different departments (A variation of the update U6)



```
U6′:    UPDATE    EMPLOYEE
        SET       Salary =
        CASE      WHEN      Dno = 5      THEN Salary + 2000
                  WHEN      Dno = 4      THEN Salary + 1500
                  WHEN      Dno = 1      THEN Salary + 3000
                  ELSE      Salary + 0 ;
```

EXPANDED Block Structure of SQL Queries



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